UNIVERSITÉ DE MONTRÉAL

CENTRE DE RECHERCHE SUR LES TRANSPORTS

CASE POSTALE 6128, SUCCURSALE A, MONTREAL, QUEBEC H3C 3J7

LA FABRICATION D'HORAIRES DE TRAVAIL POUR LES CONDUCTEURS D'AUTOBUS PAR UNE MÉTHODE DE GÉNÉRATION DE COLONNES

par

Martin Desrochers

Thèse de Ph.D., département d'informatique et de recherche opérationnelle, dirigée par François Soumis.

Cette étude a été publiée grâce à une subvention du Fonds F.C.A.R. pour l'aide et le soutien à la recherche.

Université de Montréal

Centre de recherche sur les transports - Publication #470

Juin 1986

Dépôt légal - 3e trimestre 1986

À MES PARENTS

TABLE DES MATIÈRES

Page

TABLE DES MATIÈRES. iii

LISTE DES ILLUSTRATIONS v

SOMMAIRE. viii

INTRODUCTION. 1

CHAPITRE I

LES HORAIRES DE TRAVAIL DES CONDUCTEURS D'AUTOBUS. 4

CHAPITRE II

QUELQUES MÉTHODES DE FABRICATION D'HORAIRES DE TRAVAIL POUR LES CONDUCTEURS D'AUTOBUS URBAINS 11

2.1 Les modèles de recouvrement et partionnement 12
2.2 Une méthode basée sur la relaxation lagrangienne 15
2.3 Une méthode basée sur le problème de couplage. 16
2.4 Une modélisation par programmation linéaire. 21
2.5 Conclusion 28

CHAPITRE III

LA FABRICATION D'UN HORAIRE DE TRAVAIL POUR LES CONDUCTEURS D'AUTOBUS URBAINS PAR GÉNÉRATION DE COLONNES. 29

3.1 La génération de colonnes. 29
3.2 Les horaires de travail de conducteurs d'autobus urbains. 36
3.3 La modélisation de la génération d'une journée de travail 39
3.4 La résolution du problème maître 47
3.5 Conclusion 59

CHAPITRE IV

UN ALGORITHME DE MARQUAGE GÉNÉRALISÉ POUR LE PROBLÈME DE PLUS COURT CHEMIN AVEC CONTRAINTES D'HORAIRES 60

4.1 Le problème. 60
4.2 Revue de la littérature. 61
4.3 Quelques caractéristiques de PPCCCH. 65
4.4 Un nouvel ordre de traitement des étiquettes 67
4.5 L'algorithme de Dijkstra généralisé. 71
4.6 La complexité de l'algorithme. 75
4.7 Résultats numériques 78
4.8 Conclusion 84

Page

CHAPITRE V

UN ALGORITHME DE RÉOPTIMISATION POUR LE PROBLÈME DE PLUS COURT CHEMIN AVEC CONTRAINTES D'HORAIRES. 86

5.1 Le problème. 86
5.2 Revue de la littérature. 87
5.3 Utilisation de l'ancienne solution 90
5.4 Principes de la réoptimisation 92
5.5 L'algorithme de réoptimisation 96
5.6 Complexité et implantation 99
5.7 Résultats numériques 102
5.8 Conclusion 106

CHAPITRE VI

UN ALGORITHME POUR LE PROBLÈME DU PLUS COURT CHEMIN AVEC CONTRAINTES SUPPLÉMENTAIRES 110

6.1 Le problème 110
6.2 Revue de la littérature 112
6.3 Les principes du pulling multidimentionnel. 114
6.4 Ordre de traitement des étiquettes. 119
6.5 L'algorithme. 122
6.6 Exemple numérique 123
6.7 Le choix de la zone à explorer. 126
6.8 Implantation et complexité. 128
6.9 Résultats numériques. 132
6.10 Conclusion. 138

CHAPITRE VII

LA FABRICATION D'HORAIRES DE TRAVAIL PAR GÉNÉRATION DE COLONNES : IMPLANTATION ET RÉSULTATS NUMÉRIQUES. 140

7.1 Les sous-problèmes 140
7.2 Solution initiale et sous-problèmes initiaux 142
7.3 Un problème provenant d'une ville américaine 142
7.4 Un problème provenant d'une ville anglaise 149
7.5 Conclusion 157

CONCLUSION. 158

REMERCIEMENTS 162

RÉFÉRENCES. 164

TABLEAU DES ILLUSTRATIONS

Page

LISTE DES FIGURES :

Figure 1.1 : Niveau de service. 4

Figure 1.2 : Tournée divisée par 4 points de relève en 5 tâches. 5

Figure 2.1 : Partition des noeuds en niveaux. 18

Figure 2.2 : Sous-problème à l'itération k. 18

Figure 2.3 : Nouvelles pièces partielles obtenues après l'itération k. 19

Figure 2.4 : Le sous-problème obtenu à l'itération k en libérant le premier noeud des pièces partielles les plus longues 19

Figure 2.5 : Pièces partielles obtenues après l'itération k en utilisant l'algorithme modifié. 20

Figure 2.6 : Partition d'un bloc en pièces légales. 22

Figure 3.1 : Exemple des différents type d'arcs 42

Figure 3.2 : Algorithme de résolution du problème maître. 48

Figure 4.1 : Relation de dominance entre étiquette associées à différents chemins 66

Figure 4.2 : Ordre de traitement des étiquettes selon leurs positions dans les fenêtres de temps 74

Figure 5.1 : Relation de dominance entre étiquettes associées à différents chemins 90

Figure 5.2 : Exemple des fonctions primale et duale à un noeud. . . . 92

Figure 5.3 : Effet de la taille de la zone d'exploration sur le temps d'exécution de l'algorithme de réoptimisation. . . 105

Figure 6.1 : Relation de dominance entre étiquettes associées à différents chemins 116

Figure 6.2 : Exemple de fonctions primale et duale. 117

Figure 6.3 : Initialisation et réseau 124

Figure 6.4 : Pulling de l'étiquette $(1,-\infty)$ au noeud B 125

Figure 6.5 : Pulling de l'étiquette $(3,-\infty)$ au noeud C 125

Page

LISTE DES FIGURES (suite) :

Figure 6.6 : Pulling de l'étiquette (5,-∞) au noeud D 126

Figure 6.7 : Pulling de l'étiquette (8,-∞) au noeud C 126

Figure 7.1 : Courbe de service du premier problème. 143

Figure 7.2 : Courbe de service du second problème 150

LISTE DES TABLEAUX :

Tableau 4.1 : Sommaire de la complexité des algorithmes 78

Tableau 4.2 : Comparaison du temps de gestion de la mémoire 80

Tableau 4.3 : Statistiques d'exécution des deux algorithmes 81
(1ère partie)

: Statistiques d'exécution des deux algorithmes 82
(2e partie)

Tableau 5.1 : Sommaire de la complexité pour la réoptimisation. . . . 102

Tableau 5.2 : Statistiques d'exécution des deux algorithmes
pour des problèmes de réoptimisation. 107

Tableau 5.3 : Statistiques d'exécution des deux algorithmes
pour des problèmes d'optimisation 108

Tableau 6.1 : Sommaire de la complexité des algorithmmes de
reaching et de pulling multidimensionnel. 132

Tableau 6.2 : Statistiques d'exécution des deux implantations
pour des problèmes de trois contraintes 132

Tableau 6.3 : Statistiques d'exécution des deux implantations
pour des problèmes de cinq contraintes. 136

Tableau 7.1 : Caractéristiques des noeuds étudiés par la méthode
de branchement pour le premier problème 147

Tableau 7.2 : Comparaison des caractéristiques des solutions du
premier problème. 148

Tableau 7.3 : Caractéristiques des noeuds étudiés par la méthode
de branchement pour le second problème. 153

Tableau 7.4 : Comparaison des caractéristiques des solutions du
second problème 154

Page

LISTE DES TABLEAUX (suite) :

Tableau 7.5 : Distribution de la durée du repas dans les trois solutions 155

Tableau 7.6 : Distribution de l'amplitude dans les trois solutions. . 156

SOMMAIRE

Toutes les sociétés de transport urbain font face au même problème : fabriquer des horaires de travail pour leurs conducteurs. La société doit affecter les conducteurs de façon à toujours avoir un conducteur dans chaque autobus en service tout en minimisant ses coûts d'exploitation. Cette affectation doit tenir compte de deux facteurs principaux : l'horaire de service et les contraintes et coûts sur les journées de travail contenus dans la convention collective.

Trois types d'algorithmes sont principalement utilisés pour résoudre ces problèmes : 1^0) des algorithmes découpant les tournées d'autobus en pièces de travail, assemblant les pièces pour former des journées de travail et faisant des échanges de pièces entre journées pour améliorer l'horaire 2^0) la méthode de recouvrement où un problème de recouvrement est utilisé pour choisir les colonnes (journées de travail) formant l'horaire de coût minimum 3^0) la méthode HASTUS où, dans une première phase, un modèle mathématique construit un horaire approximatif et, dans une deuxième phase, les pièces de travail sont découpées et un horaire définitif est construit.

Depuis plusieurs années, la méthode de génération de colonnes a été utilisée pour résoudre des problèmes de routing. Cette méthode décompose l'obtention d'une solution en deux problèmes. Le problème-maître, un problème de recouvrement dont les colonnes sont des journées de travail, choisit la solution à coût minimum. Le sous-problème fabrique la colonne de coût marginal minimum. Si son coût marginal est négatif, la colonne est ajoutée au problème-maître sinon la solution du problème-maître est optimale.

Nous voulons vérifier si cette méthode permet d'améliorer les horaires construits par les méthodes actuelles. Nous avons modélisé le sous-problème de fabrication de journées de travail comme un problème de plus court chemin avec contraintes supplémentaires.

L'étude de la résolution des problèmes de plus court chemin avec contraintes supplémentaires représente la contribution théorique de cette thèse.

La première partie de l'étude portait sur le problème de plus court chemin avec contraintes d'horaires. Nous avons développé le premier algorithme pseudo-polynomial pour le résoudre, en utilisant une procédure de marquage des successeurs (reaching) et un ordre de traitement des étiquettes. De plus, nous avons développé un algorithme pseudo-polynomial pour la réoptimisation après l'augmentation du coût de certains arcs, en utilisant une procédure de marquage par les prédecesseurs (pulling). Cet algorithme par pulling s'est avéré plus efficace que l'algorithme par reaching tant au niveau de la complexité que du temps d'exécution.

La seconde partie de l'étude portait sur le problème de plus court chemin avec plusieurs contraintes supplémentaires. Nous avons généralisé l'algorithme utilisant le marquage par les prédécesseurs (pulling) au cas de problèmes multidimentionnels. L'étude de la complexité de l'algorithme de pulling et de celle d'un algorithme de Dijkstra généralisé permet de conclure à la supériorité de notre implantation de l'algorithme de pulling.

Finalement, la méthode de génération de colonnes est utilisée pour solutionner deux problèmes-tests. Le premier problème provient d'une ville américaine et comporte 167 tâches. La méthode de génération de colonnes a permis d'améliorer le coût de la meilleure solution connue de 0,7%. Le second problème provient d'une ville anglaise et comporte 235 tâches. La

méthode de génération de colonnes a permis d'améliorer le coût de la meilleure solution connue de 2,2%. Même si ces améliorations peuvent sembler mineures, elles peuvent représenter de grandes sommes. Par exemple, à Montréal, le budget annuel de rémunération des 4 400 conducteurs était de 165 millions de dollars en 1985.

Bien qu'il reste encore du travail avant l'obtention d'une version commercialisable de la méthode de génération de colonnes pour la fabrication d'horaires de travail pour les conducteurs d'autobus urbains, ces résultats confirment l'intérêt de cette méthode.

INTRODUCTION

Depuis la fin des années 60, beaucoup d'efforts ont été consacrés à la mise au point de méthodes de fabrication d'horaires de travail pour les conducteurs d'autobus urbains. Ces efforts s'expliquent par l'importance et l'universalité du problème. En effet, presque toutes les villes importantes ont un service d'autobus et les exploitants doivent fabriquer de nouveaux horaires de travail à chaque changement dans le service. De plus, puisqu'un horaire de travail est utilisé répétitivement, diminuer le coût d'un horaire de 1% représente déjà une somme importante. Toutes ces raisons expliquent que la mise au point d'algorithmes pour fabriquer ces horaires ait à son tour une importance économique.

La première génération d'algorithmes tentait de reproduire en les automatisant les méthodes de fabrication d'horaires des répartiteurs à l'emploi des compagnies de transport urbain. Ces méthodes procédaient par découpage des routes d'autobus en pièces de travail et en combinant les pièces obtenues pour former des journées de travail. Les horaires étaient améliorés en modifiant le découpage des pièces ou en échangeant des pièces entre deux journées. Comme tous les algorithmes d'échanges, ces méthodes ont tendance à trouver un optimum local.

La seconde génération d'algorithmes est caractérisée par l'application de la programmation mathématique. Au chapitre II, nous verrons en détail plusieurs méthodes de fabrication d'horaires utilisant la programmation mathématique. La plupart de ces méthodes divisent la fabrication d'un horaire en plusieurs phases sans utiliser une des méthodes de décomposition connues et ne peuvent être utilisées pour obtenir une solution optimale. Par exemple, la méthode de recouvrement énumère dans une première phase les journées de travail et ne conserve que celles "ayant le plus de possibilités" de faire partie de la solution optimale. Généralement, les journées de

travail retenues sont semblables aux journées déjà utilisées. Cette première phase limite l'efficacité de la méthode de recouvrement et ne permet pas l'obtention d'une bonne solution lors d'une grande modification des conditions.

Dans cette thèse, nous utilisons une méthode ayant depuis quelques années permis de résoudre des problèmes complexes de routing : la génération de colonnes. Cette méthode est un cas particulier de la décomposition de Dantzig-Wolfe [DESR 84a]. La fabrication d'un horaire est divisée en deux phases : la résolution du problème-maître déterminant les journées faisant partie de l'horaire et la résolution de sous-problèmes fabriquant des journées de travail permettant d'améliorer l'horaire actuel du problème-maître. Celui-ci est un problème de recouvrement assurant qu'en tout temps il y ait un conducteur dans chaque autobus. L'utilisation d'une méthode de génération de colonnes demande la définition et la résolution d'un sous-problème de fabrication de journées de travail. Au chapitre III, la fabrication d'une journée de travail est modélisée comme un problème de plus court chemin avec contraintes supplémentaires.

Les chapitres subséquents traitent de la résolution de ces problèmes. Le chapitre IV présente une généralisation de l'algorithme de Dijkstra pour résoudre les problèmes de plus court chemin avec une contrainte supplémentaire. Un nouvel algorithme de réoptimisation pour le problème de plus court chemin avec une contrainte supplémentaire est exposé au chapitre V. En plus de permettre la réoptimisation, ce nouvel algorithme peut être utilisé pour résoudre ces problèmes. L'adaptation du nouvel algorithme pour des problèmes de plus court chemin avec plusieurs contraintes supplémentaires est présentée au chapitre VI. Cet algorithme permet la fabrication de journées de travail selon le modèle exposé au chapitre III.

Le but principal de cette thèse est de démontrer que l'approche de génération de colonnes peut produire de meilleurs résultats que les

méthodologies actuelles. Nous pourrons ainsi démontrer le potentiel de cette approche avant de poursuivre le développement d'un logiciel. Le dernier chapitre de la thèse effectue une telle comparaison pour deux problèmes-tests provenant l'un d'une ville américaine et l'autre d'une ville anglaise. La résolution de ces deux problèmes et la comparaison des solutions obtenues permettent de déterminer le potentiel de la méthode et de nouvelles directions de recherche.

Plusieurs chapitres de cette thèse ont été rédigés de façon à pouvoir être lus indépendamment. Ceci implique un certain niveau de redondance entre les chapitres. Nous croyons que les inconvénients que cela pourrait poser au lecteur sont compensés par la possibilité de pouvoir lire indépendamment chaque chapitre.

CHAPITRE I

LES HORAIRES DE TRAVAIL DES CONDUCTEURS D'AUTOBUS

Les sociétés de transport urbain ont pour mandat de satisfaire les besoins en transport de la population. Pour ce faire, elles doivent prendre plusieurs niveaux de décisions. Au niveau des décisions stratégiques, la société doit planifier le réseau de service offert i.e. lignes d'autobus, de métro, arrêts, etc. Au niveau tactique, la société décide du niveau de service (voir figure 1.1) offert sur chaque ligne pour répondre aux besoins du public. L'ensemble des tournées d'autobus compose l'horaire associé au niveau de service choisi. Chaque tournée comprend une heure de départ du garage, une durée de service sur une ou plusieurs lignes et une heure de retour au garage. L'ensemble des tournées d'autobus compose l'horaire de service offert au public.

Le problème qui nous intéresse : la fabrication d'horaires de travail pour les conducteurs d'autobus, constitue le dernier niveau de décision. La société doit affecter les conducteurs de façon à toujours avoir un conducteur dans chaque autobus en service. Cette affectation doit tenir compte de deux facteurs principaux : l'horaire de service et les contraintes sur les journées de travail contenues dans la convention collective.

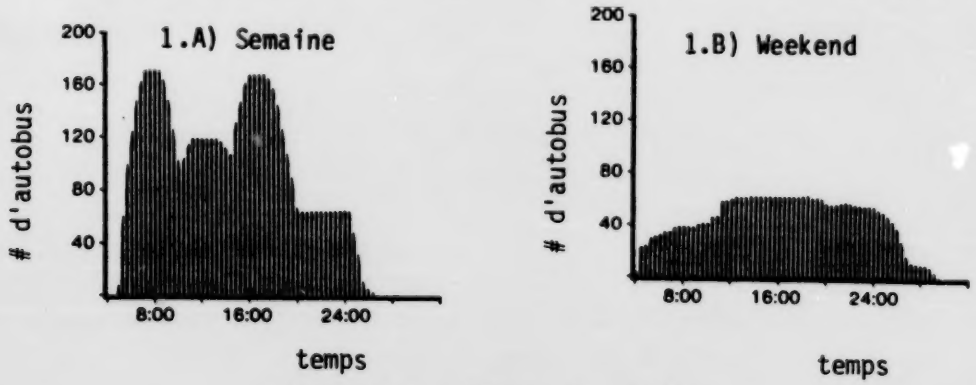


Figure 1.1 : Niveau de service

Nombre de tournées d'autobus en service selon l'heure [CARR 82]

Avant de poursuivre, définissons d'abord quelques termes relatifs à la fabrication d'horaires de conducteurs d'autobus.

tournée : trajet d'un véhicule entre son départ et son retour au garage; le terme "bloc" est souvent utilisé pour désigner une tournée;

point de relève : endroit où un conducteur peut en remplacer un autre dans une tournée;

tâche : partie d'une tournée comprise entre deux points de relève consécutifs;

pièce de travail : suite de tâches consécutives d'une même tournée effectuée par un même conducteur.

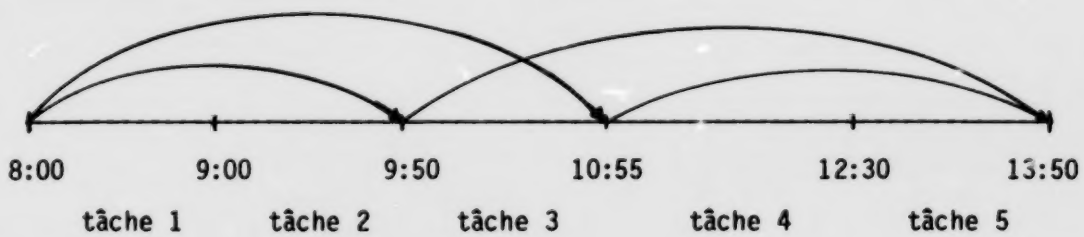


Figure 1.2 : Tournée divisée par 4 points de relève en 5 tâches. Les arcs représentent des pièces de travail possibles.

journée de travail : une ou plusieurs pièces de travail effectuées par le même conducteur;

pause : période d'inactivité du conducteur comprise entre deux pièces de travail;

journée continue : journée de travail composée d'une seule pièce; dans certaines conventions collectives, les journées composées de deux pièces et d'une très courte pause sont considérées comme des journées continues;

journée brisée : journée de travail de 2 pièces ou plus (et qui n'est pas considérée comme une journée continue);

amplitude : *différence entre l'heure du début de la première pièce* et la fin de la dernière pièce d'une journée de travail;

pièce vacante : petite pièce de travail non affectée à une journée de travail.

La fabrication d'horaires de travail pour les conducteurs d'autobus vise à recouvrir par des journées de travail au moindre coût toutes les tâches. La convention collective réglemente les salaires, les primes, le découpage des tournées en pièces de travail, les caractéristiques des journées de travail et des contraintes globales sur toutes les journées de travail (pourcentage minimum de journées continues, etc.).

Le logiciel HASTUS [BLAIS 76, LESS 81, ROUS 85b] est utilisé par plusieurs sociétés de transport urbain pour fabriquer les horaires de travail des conducteurs. Lors de l'utilisation de HASTUS, l'utilisateur décrit la convention collective à l'aide d'une série de paramètres [GIRO 83]. Ces paramètres permettent de décrire la majorité des conventions collectives. La description des principaux paramètres permet d'introduire le lecteur aux différentes caractéristiques et contraintes possibles.

A) découpage des tournées (blocs)

MAX-RÉALISABILITÉ ℓ : il est interdit de découper en plusieurs pièces une tournée de durée inférieure à ℓ.

PIÈCE-LG-MIN
PIÈCE-LG-MAX : durée minimale et maximale d'une pièce de travail.

De plus, il peut être interdit de découper les tournées durant certaines périodes, ex. : durant les périodes de pointe. Le coût d'une pièce de travail dépend de sa durée.

COÛT-HORAIRE : taux horaire d'un conducteur.

Toutefois, il est possible que pour les petites pièces de travail, la société garantisse une rémunération minimale pour chaque pièce.

GARANTI-PIÈCE h : durée minimale garantie pour une pièce de travail. Pour les pièces de travail durant moins de h minutes, la société paie h minutes de travail.

B) allocations et pauses

En plus du travail du conducteur dans un véhicule en service, un certain nombre d'autres éléments font partie des pièces de travail, ex. : au début (ou à la fin) de chaque pièce, le conducteur peut avoir à se déplacer pour rejoindre son véhicule (ou retourner au garage), ce déplacement fait généralement partie de la pièce de travail.

PREPAR-SORTIE : temps de préparation en début de journée.

PREPAR-RENTRÉE : temps de rapport à la fin de la journée.

Les pauses peuvent être payées ou non selon la convention collective :

TAUX-PAUSE-BRIS : portion du taux horaire payée pour les pauses des journées brisées.

TAUX-PAUSE-CONTINUE : portion du taux horaire payée pour les pauses des journées continues.

Pour les journées de trois pièces, trois cas sont possibles :

1) aucune pause payée
2) la plus petite pause est payée
3) la somme des pauses est payée à un taux horaire prédéterminé (ex. : demi-temps).

C) caractéristiques d'une journée de travail

PIÈCE-NB-MAX : nombre maximum de pièces de travail dans une journée.

JOURNÉE-LG-MIN
JOURNÉE-LG-MAX : durées travaillées minimale et maximale pour une journée.

AMPLITUDE-MAXIMUM : amplitude maximale permise pour une journée de travail.

Les bornes minimale et maximale sur la durée travaillée d'une journée de travail encadrent la durée souhaitée de travail. Si un conducteur travaille moins que cette durée souhaitée, l'employeur peut lui garantir une rémunération minimale pour une journée de travail.

GARANTIE-JOURNÉE h : temps minimum garanti pour une journée de travail.

Toutefois si un conducteur travaille plus que la durée souhaitée, l'employeur peut devoir lui payer un supplément pour chaque heure travaillée en plus.

COÛT-SUPPL-TAUX : taux à appliquer au salaire horaire pour obtenir le coût horaire en temps supplémentaire.

Les journées de travail ayant une grande amplitude ne sont pas appréciées des conducteurs. Les conventions collectives comprennent presque toujours des coûts supplémentaires associés à ces journées.

AMPLITUDE-BONI h b : le taux-horaire est augmenté de b pour la partie de journée dépassant l'amplitude h.

AMPLITUDE-TAUX h t : le taux-horaire est multiplié par t pour la partie de journée dépassant l'amplitude h.

D) <u>caractéristiques particulières des journées continues</u>

CONTINUE-LG-MIN
CONTINUE-LG-MAX : durées travaillées minimale et maximale pour une journée continue.

PAUSE-MIN-CONTINUE
PAUSE-MAX-CONTINUE : certaines conventions collectives considèrent les journées de 2 pièces avec une courte pause comme une journée continue.

Les pourcentages minimum et maximum du nombre de journées continues sont souvent réglementés par la convention collective

MIN1P
MAX1P : pourcentages minimum et maximum du nombre de journées continues dans l'horaire.

E) caractéristiques particulières des journées brisées

Les journées brisées sont utilisées pour offrir le service en période de pointe. Les journées brisées ne sont pas toujours appréciées des conducteurs. Dans le cas des journées brisées de plus grande amplitude, la convention collective prévoit souvent des primes pour rendre ces journées acceptables aux conducteurs.

BRIS-LG-MIN
BRIS-LG-MAX : durées travaillées minimale et maximale pour une journée brisée.

PAUSE-MIN-BRIS
PAUSE-MAX-BRIS : durées minimale et maximale des pauses dans les journées brisées. Parfois la convention collective n'impose une durée minimale que sur la plus longue des pauses.

F) caractéristiques particulières des pièces vacantes

Les pièces vacantes sont utilisées pour recouvrir les tâches non-affectées à des journées de travail. Généralement les pièces vacantes sont effectuées en temps supplémentaire après ou avant la journée régulière d'un conducteur.

GARANTI-VACANT : durée minimale garantie pour une pièce vacante.

TAUX-VACANT : taux à appliquer au salaire horaire pour obtenir le coût horaire des pièces vacantes.

Les pièces vacantes étant plutôt onéreuses, les employeurs cherchent à limiter leur nombre.

NB-VACANT : nombre de pièces vacantes permises.

G) contraintes globales sur l'amplitude

Les conventions collectives contiennent parfois une ou des contraintes limitant le pourcentage du nombre de journées de travail ayant une certaine amplitude.

CONTRAINTE AMPLITUDE A1 [OU PLUS / OU MOINS] [MINIMUM / MAXIMUM] P2

Cette contrainte limite le nombre de journées ayant A1 ou plus (ou moins) d'amplitude à au minimum (au maximum) P2 pourcent du nombre de journées de l'horaire.

La notation du logiciel HASTUS sera utilisée pour décrire les conventions collectives dans les chapitres subséquents.

CHAPITRE II

QUELQUES MÉTHODES DE FABRICATION D'HORAIRES DE TRAVAIL POUR LES CONDUCTEURS D'AUTOBUS URBAINS

Presque toutes les villes ont un service d'autobus urbains et doivent fabriquer des horaires de travail. Depuis le début des années 70, plusieurs logiciels ont été mis au point pour automatiser la fabrication de ces horaires. Les premiers logiciels mis au point sont : COMPACS à l'Université de Leeds sous la direction de A. WREN [MANI 75] et RUCUS aux États-Unis par la compagnie MITRE [WILH 75]. Ces deux logiciels fabriquent des horaires en deux phases. Lors de la première phase, les blocs sont découpés en pièces et des journées de travail réalisables sont créées. Lors de la seconde phase, l'horaire est amélioré en modifiant le découpage des blocs en pièces et en échangeant les pièces entre les journées de travail. Ces méthodes d'échanges ne permettent que d'optimiser localement les horaires de travail.

Plus tard, la mise au point de nouvelles formulations mathématiques a permis le développement de logiciels utilisant diverses méthodes de programmation mathématique et optimisant globalement les horaires. Dans ce chapitre nous exposerons les trois principales formulations mathématiques proposées : le recouvrement, le couplage et une modélisation par programmation linéaire. Les lecteurs intéressés par les autres approches pourront consulter les comptes rendus des deux derniers symposiums sur la fabrication d'horaires pour les véhicules et conducteurs dans les transports publics [WREN 81, ROUS 85a] . Ces volumes contiennent les principaux articles présentés lors des symposiums tenus à LEEDS (1980) et Montréal (1983). De plus, il existe trois revues de la littérature [BODI 83, CARR 84, ROUS 84].

Bodin et al. [BODI 83] effectuent un rapide survol du domaine dans leur grand survol du routing et de l'ordonnancement. Carraresi et Gallo

[CARR 84] exposent les méthodes de fabrication d'horaires utilisant des modèles de réseaux. Rousseau [ROUS 84] expose plusieurs méthodes de fabrication d'horaires de travail tant pour les conducteurs d'autobus que pour d'autres types d'employés.

2.1 LES MODÈLES DE RECOUVREMENT ET DE PARTITIONNEMENT

Les modèles de recouvrement et de partitionnement ont été utilisés depuis les années 60 pour modéliser la fabrication des horaires de travail des équipages d'avions [ARAB 69]. Ces modèles ont été repris pour la fabrication d'horaires de travail pour les conducteurs d'autobus urbains.

$$\text{Min} \sum_{j \in J} c_j x_j \qquad (2.1)$$

sous les contraintes

$$\sum_{j \in J} a_{ij} x_j \geq (=)\ 1 \qquad \text{pour tout } i \in I \qquad (2.2)$$

$$\sum_{j \in J} d_{kj} x_j \leq D_k \qquad k=1,\ldots,K \qquad (2.3)$$

$$x_j = \{0,1\} \qquad \text{pour tout } j \in J \qquad (2.4)$$

avec

I : l'ensemble des tâches i composant les blocs.

J : l'ensemble des journées de travail réalisables.

x_j : $\begin{cases} 1 & \text{si la journée } j \text{ appartient à la solution.} \\ 0 & \text{sinon} \end{cases}$

c_j : coût de la journée j.

a_{ij} : $\begin{cases} 1 & \text{si la journée } j \text{ recouvre la tâche } i. \\ 0 & \text{sinon} \end{cases}$

K : nombre de contraintes additionnelles où $d_{kj} x_j \leq D_k$ est une contrainte linéaire quelconque.

Le modèle de recouvrement (de partitionnement) peut aussi comprendre des contraintes additionnelles (2.3). Par exemple, une contrainte peut limiter le nombre de journées brisées

$$d_{1j} = \begin{cases} 1 & \text{si la journée j est brisée.} \\ 0 & \text{sinon} \end{cases}$$

D_1 = nombre maximum de journées brisées.

Idéalement l'ensemble J devrait contenir toutes les journées de travail réalisables. À cause du grand nombre de ces journées, on utilise des règles heuristiques pour réduire la dimension du problème. Une première réduction consiste à réduire le nombre de variables en éliminant toutes les journées de travail réalisables inefficaces, i.e. n'ayant aucune probabilité d'être dans une solution. Une seconde réduction consiste à diminuer le nombre de tâches en éliminant les points de relève ayant peu de chance d'être utilisés. L'élimination d'un point de relève permet la fusion des deux tâches adjacentes. Cette réduction permet une réduction du nombre de contraintes et de variables. Ces réductions sont gouvernées par des paramètres fixés en utilisant l'expérience des répartiteurs.

Plusieurs méthodes ont été mises au point pour résoudre ce modèle. Heurgon [HEUR 72] résoud la formulation de partitionnement en utilisant la programmation linéaire et un branchement sur les variables pour obtenir une solution entière. Wren et al. [MANI 77, WREN 85] résolvent la formulation de recouvrement en utilisant la programmation linéaire mais utilisent une méthode de branchement développée par Ryan et Forster [RYAN 81] pour obtenir une solution entière.

La résolution du modèle de recouvrement permet la fabrication d'horaires de travail dont la solution comporte jusqu'à une centaine de journées. Il s'agit de problèmes de petite taille.

Ryan et Forster ont constaté que dans les méthodes de branchement sur les variables, les branches $x_j = 1$ et $x_j = 0$ n'ont pas la même force. La branche fixant une variable à 1 est beaucoup plus restrictive que celle fixant la variable à 0. Ils ont développé un branchement plus équilibré.

Soit une paire de tâches i_1 et $i_2 \in I$ avec

$$0 < \sum_{j \in J(i_1,i_2)} x_j < 1$$

où $J(i_1,i_2)$ est l'ensemble de toutes les journées de travail recouvrant simultanément les tâches i_1 et i_2, Ryan et Forster utilisent les contraintes suivantes pour définir le branchement :

$$\text{branche - 0} \qquad \sum_{j \in J(i_1,i_2)} x_j \leq 0$$

$$\text{branche - 1} \qquad \sum_{j \in J(i_1,i_2)} x_j \geq 1$$

Les expériences numériques confirment que la structure de l'arbre de branchement est plus équilibrée.

Mitra et Darby-Dowman [MITR 85] modifient le modèle de partitionnement pour obtenir un modèle de partitionnement généralisé. Ce nouveau modèle est obtenu en remplaçant les équations (2.1,2.2) par les équations suivantes :

$$\text{Min} \sum_{j \in J} c_j x_j + \sum_{i \in I} w_i^u u_i + \sum_{i \in I} w_i^o o_i \qquad (2.6)$$

$$\sum_{j \in J} a_{ij} x_j + u_i - o_i = 1 \qquad i \in I \qquad (2.7)$$

$$u_i = \{0,1\} \qquad i \in I$$

$$o_i \geq 0 \quad \text{et entier} \qquad i \in I$$

où $u_i = \begin{cases} 1 & \text{si la tâche i n'est pas recouverte.} \\ 0 & \text{sinon} \end{cases}$

o_i = nombre de surrecouvrement de la tâche i.

w_i^u = pénalité associée au non-recouvrement de la tâche i.

w_i^o = pénalité associée au surrecouvrement de la tâche i.

Les auteurs utilisent un logiciel adapté de programmation linéaire en nombres entiers pour résoudre le modèle.

2.2 UNE MÉTHODE BASÉE SUR LA RELAXATION LAGRANGIENNE

Shepardson et Marsten [SHEP 80] résolvent le modèle de partitionnement pour le cas où toutes les journées de travail sont composées de deux pièces de travail. Dans un problème de partionnement, une pièce de travail est représentée dans la matrice A sous forme d'une série de 1 consécutifs dans une colonne.

Le problème se formule de la façon standard

$$\text{Min } cx$$
$$\text{s.a.} \quad Ax = 1$$
$$x_j \in \{0,1\}$$

Pour les problèmes de partionnement où toutes les journées de travail n'ont qu'une seule pièce, il est possible de transformer ce problème en problème de plus court chemin. Le découpage de chaque bloc en journées (pièces) réalisables s'effectue en résolvant le plus court chemin suivant :

$$\text{Min} \sum_{\substack{i \in N_k \\ j \in N_k}} c_{ij} x_{ij}$$

$$\sum_{j \in N_k} x_{ij} - \sum_{i \in N_k} x_{ji} = \begin{cases} 1 & \text{pour le début } p_k \text{ du bloc k.} \\ 0 & i \in N_k,\ i \neq p_k,\ i \neq q_k. \\ -1 & \text{pour la fin } q_k \text{ du bloc k.} \end{cases}$$

$$x_{ij} \in \{0,1\}$$

N_k : ensemble des points de relève du bloc k.

c_{ij} : coût de la journée débutant au point de relève i et terminant au point de relève j.

$$x_{ij} = \begin{cases} 1 & \text{si la journée débutant au point de relève i et terminant au point j est utilisée.} \\ 0 & \text{sinon.} \end{cases}$$

Cette formulation du problème avec journées d'une pièce suggère une relaxation pour le problème avec journées de deux pièces. Les colonnes x_j composées de 2 pièces sont divisées en demi-journées x_j^1 et x_j^2, composées chacune d'une pièce. Les éléments non-nuls de la matrice A sont séparés entre les matrice A^1 et A^2 où A^1 représente les premières pièces des journées et A^2 les secondes. Le coût de chaque journée c_j est séparé selon $c_j^1 + c_j^2 = c_j$. Le problème avec journée de 2 pièces se formule maintenant :

$$\begin{aligned} & \text{Min } C^1X^1 + C^2X^2 \\ \text{s.a.} \quad & \left.\begin{aligned} A^1X^1 + A^2X^2 &= 1 \\ X^1, X^2 &\in \{0,1\} \end{aligned}\right\} \quad (X^1, X^2) \in G \qquad (P) \\ & IX^1 - IX^2 = 0 \end{aligned}$$

En relaxant la contrainte $X^1 = X^2$, nous pouvons utiliser la méthode du plus court chemin pour résoudre le problème relaxé. Les auteurs construisent une relaxation Lagrangienne de (P) en dualisant les contraintes $X^1 = X^2$:

$$\begin{aligned} W(V) &= \text{Min } C^1X^1 + C^2X^2 + U(X^1 - X^2), \quad (X^1, X^2) \in G \\ &= \text{Min } (C^1+U)X^1 + (C^2 - U)X^2, \quad (X^1, X^2) \in G. \end{aligned}$$

Shepardson et Marsten utilisent une méthode de sous-gradient pour ajuster les multiplicateurs U. Une solution réalisable est obtenue par une méthode de branchement.

Cette méthode a permis la résolution de problèmes comprenant jusqu'à 156 tâches.

2.3 UNE MÉTHODE BASÉE SUR LE PROBLÈME DE COUPLAGE

Ball, Bodin et Dial [BALL 83] ont proposé une méthode heuristique basée sur une formulation de couplage. Cette méthode fabrique une solution à la fois au problème de fabrication des blocs d'autobus et au problème de fabrication des horaires de travail des conducteurs.

La méthode de fabrication utilise un réseau où les noeuds sont les points de relève et comprenant 2 types d'arcs :

1) des arcs de travail représentant un mouvement du conducteur et de son véhicule. Ces arcs représentent les tâches et les mouvements hors-service du véhicule.

2) des arcs de transitions où le conducteur quitte son véhicule pour pour un autre ou pour retourner au garage. Ces arcs représentent des allocations et des pauses.

Un gros problème de fabrication d'horaires des véhicules et des conducteurs peut comprendre jusqu'à 6000 arcs de travail et plus d'un million d'arcs de transition. Ces dimensions rendant impossible la fabrication des horaires en une seule phase. L'algorithme de Ball et al. débute par la fabrication de pièces de travail en créant des chemins d'arcs de service de longueur inférieure à T où T représente la longueur maximale d'une pièce. L'algorithme améliore ensuite les pièces de travail et finalement assemble les pièces pour former des journées de travail.

La première phase, la génération des pièces de travail, débute par la partition des noeuds du réseau en niveaux. Le niveau d'un noeud i est le nombre maximum d'arcs dans un chemin de la source au noeud i (figure 2.1). L'algorithme génère itérativement des pièces de plus en plus longues. À l'itération k, les noeuds du niveau k sont ajoutés au réseau et l'algorithme tente de les fusionner aux pièces partielles déjà existantes. Si un noeud ne peut être fusionné à une pièce partielle, il devient le début d'une nouvelle pièce de travail.

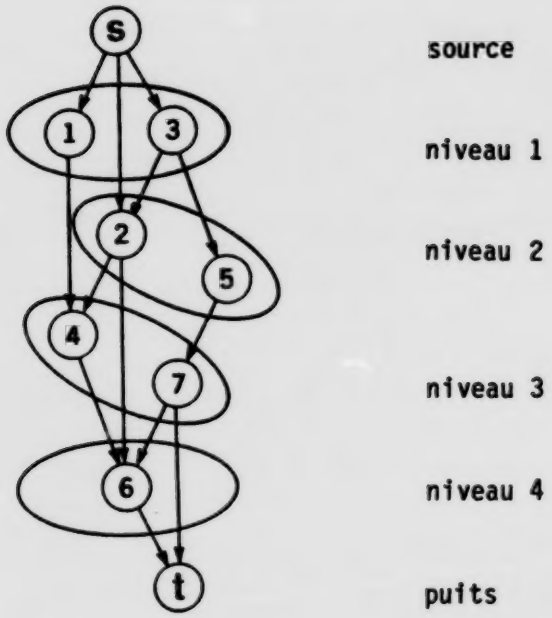


Figure 2.1 : Partition des noeuds en niveaux [BALL 83].

La figure 2.2 illustre le sous-problème à l'itération k. Les noeuds {13,14,15,16} du niveau k sont ajoutés au réseau. L'ensemble optimal de fusions de noeuds est obtenu en résolvant le problème de couplage de coût minimum de la figure 2.2. Dans ce réseau, un arc entre une fin de pièce partielle et un noeud existe si la pièce résultante est légale. Dans la figure 2.2, les arcs du couplage sont marqués et la figure 2.3 illustre les nouvelles pièces partielles obtenues.

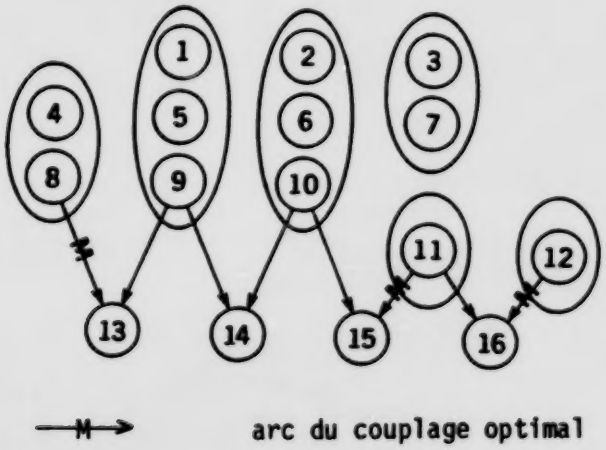


Figure 2.2 : Sous-problème à l'itération k [BALL 83].

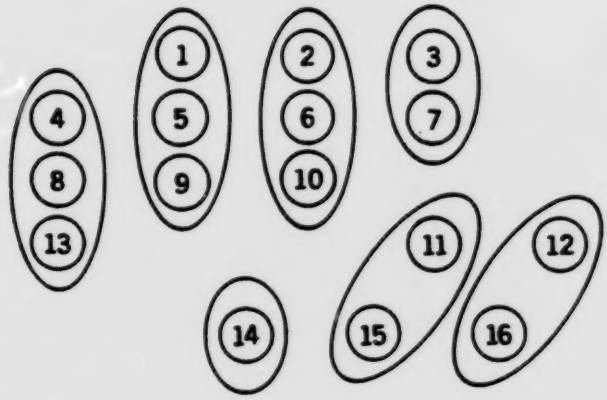
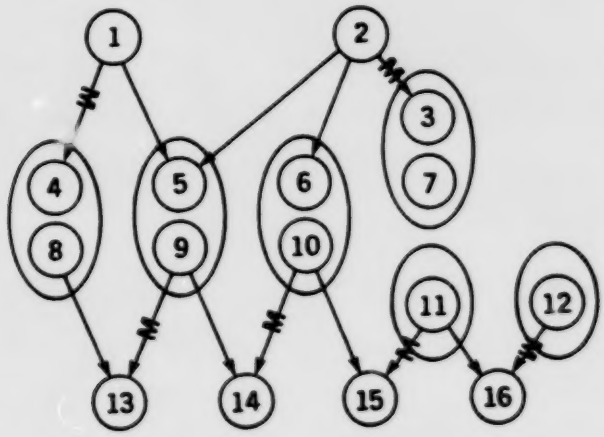


Figure 2.3 : Nouvelles pièces partielles obtenues après l'itération k.

L'efficacité de la première phase peut être améliorée, si lorsque la longueur des pièces partielles est presque égale à la longueur maximale T d'une pièce, nous libérons le premier noeud de chaque pièce. Ceci permet à l'algorithme de couplage de choisir entre la fusion avec le début actuel de la pièce et la fusion avec une pièce de niveau k. La présence d'un arc entre 2 noeuds indique que leur fusion est légale. La figure 2.4 illustre ce sous-problème à l'itération k et les arcs faisant partie du couplage optimal sont marqués. La figure 2.5 illustre les pièces partielles obtenues à la fin de l'itération k.



: arcs faisant partie du couplage optimal

Figure 2.4 : Le sous-problème obtenu à l'itération k en libérant le premier noeud des pièces partielles les plus longues.

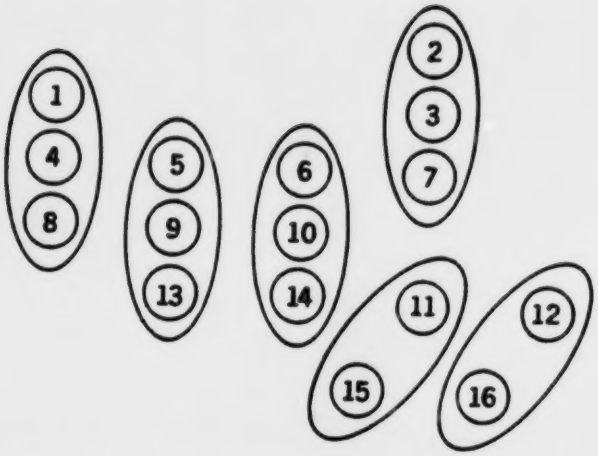


Figure 2.5 : Pièces partielles obtenues après l'itération k en utilisant l'algorithme modifié.

La seconde phase considère la possibilité de briser chaque pièce formée et de la recombiner avec toutes les autres pièces. Un réseau est construit, les noeuds sont les pièces et chaque arc est associé à une combinaison légale de pièces. Le coût sur chaque arc est déterminé de façon à minimiser le nombre de pièces et à défavoriser les pièces de faible durée. Les nouvelles pièces sont obtenues par la résolution d'un problème de couplage de coût minimum.

La troisième phase combine les pièces de travail pour fabriquer des journées de travail. Cette phase peut aussi être formulée comme un problème de couplage. Le coût d'un arc (i,j) est le coût de la journée de travail composée des pièces i et j.

Cette méthode n'a pas été utilisée par une compagnie de transport urbain mais a permis de fabriquer des horaires pour des problèmes d'environ 140 journées de travail. La principale faiblesse de cette méthode est la définition des coûts sur les arcs lors des deux premières phases. Lors de la troisième phase, cette faiblesse disparaît. Cette méthode fabrique les meilleures journées possibles pour un ensemble de pièces fabriquées heuristiquement lors des deux premières phases.

2.4 UNE MODÉLISATION PAR PROGRAMMATION LINÉAIRE

Les méthodes de résolution déjà exposées permettent de résoudre des problèmes comportant au plus jusqu'à 150 journées de travail. Ces méthodes ne peuvent résoudre les problèmes de grande dimension qui peuvent comporter jusqu'à 350 journées de travail. Le logiciel HASTUS a été développé pour résoudre de tels problèmes. BLAIS et al. [BLAI 76] ont développé un modèle mathématique utilisant la programmation linéaire pour décrire les horaires de travail. À cause de sa grande taille, cette formulation ne peut être résolue en une seule phase. Les diverses phases du logiciel HASTUS [LESS 81, ROUS 85b] permettent la résolution heuristique de cette formulation.

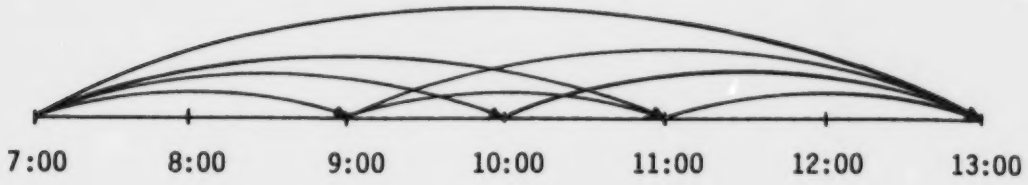
Le modèle HASTUS

Pour simplifier la présentation du modèle, nous assumons que les journées de travail sont composées de deux pièces de travail. Toutefois, le modèle permet de considérer les journées de plus de 2 pièces.

Chaque couple ij représente une pièce de travail débutant au temps i et terminant au temps j. Seuls les couples ij correspondant à une pièce réalisable sont considérées. Les temps i et j correspondent à un début de bloc, un point de relève ou une fin de bloc pour un bloc donné. La figure 2.6 illustre les partitions légales d'un bloc en pièces. Pour être légale une pièce doit respecter les contraintes de la convention collective sur les pièces et faire partie d'une partition légale d'un bloc. Le couple 00 représente une pièce de travail inexistante et est utilisé pour la représentation des journées d'une pièce.

Un quadruplet ijkh représente une journée de travail composée des pièces représentées par les couples ij et kh. Seules les journées de travail respectant la convention collective sont considérées. Si ij ou kh est une pièce inexistante 00, la journée est soit une journée d'une pièce, soit une pièce vacante. Nous dénoterons journée ijkh, la journée de travail

représentée par le quadruplet ijkh, et pièce ij, la pièce de travail représentée par le couple ij.



Les seules pièces légales sont

7-9	7-11	9-11	10-13
7-10	7-13	9-13	11-13

Figure 2.6 : Partition d'un bloc en pièces légales [LESS 81]

Définissons

L : l'ensemble des blocs p.

K : l'ensemble des pièces légales ij.

I : l'ensemble des journées réalisables ijkh.

I(ij) $\subset$ I : l'ensemble des journées réalisables comprenant la pièce ij; ij00 fait partie de I(ij).

T_p : l'ensemble des temps correspondant au début, à la fin ou à un point de relève du bloc p.

x_{ijkh} : $\begin{cases} 1 & \text{si la journée ijkh est affectée à un conducteur.} \\ 0 & \text{sinon.} \end{cases}$

c_{ijkh} : coût de la journée de travail ijkh.

y_{ij} : $\begin{cases} 1 & \text{si la pièce ij fait partie d'une journée } x_{ijkh} \text{ affectée à un conducteur.} \\ 0 & \text{sinon.} \end{cases}$

N.B. Nous supposons qu'une pièce ij n'est légale que pour un seul bloc p. Ceci évite l'utilisation des notations y^p_{ij} et x^p_{ijkh}.

Le problème de la fabrication d'horaires se formule ainsi :

$$\text{Min} \sum_{ijkh \in I} c_{ijkh} x_{ijkh} \qquad (2.8)$$

sujet à

$$\sum_{mnkh \in I(ij)} x_{mnkh} - y_{ij} = 0 \qquad \text{pour tout } ij \in K-\{00\} \qquad (2.9)$$

$$Dx \leq d \qquad (2.10)$$

$$\sum_{i \in T_p} y_{ik} - \sum_{j \in T_p} y_{kj} = b_k^p \quad \text{pour tout } k \in T_p, \text{ pour tout } p \in L \qquad (2.11)$$

$$\text{où } b_k^p = \begin{cases} -1 & \text{si k est le début du bloc p} \\ +1 & \text{si k est la fin du bloc p} \\ 0 & \text{sinon} \end{cases}$$

$$x_{ijkh} \in \{0,1\} \qquad \text{pour tout } ijkh \in I \qquad (2.12)$$

$$y_{ij} \in \{0,1\} \qquad \text{pour tout } ij \in K. \qquad (2.13)$$

Les contraintes (2.11) sont des contraintes de conservation de flot utilisées pour garantir la partition du bloc p en pièces légales. Elles forcent la ciculation d'une unité de flot du début d'un bloc à sa fin dans un réseau du type illustré à la figure 2.6.

Les contraintes (2.9) garantissent que chaque pièce de travail ij utilisée pour partitionner un bloc sera recouverte dans une journée de travail ijkh ou mnij.

Les contraintes (2.10) représentent les contraintes additionnelles de la convention collective. Elles sont du même type que les contraintes (2.3). Par exemple, elles peuvent limiter

- le nombre minimum de journées continues
- le nombre maximum de journées de travail
- la longueur moyenne d'une journée de travail
- etc.

Cette formulation ne permet pas la résolution facile de problèmes de grande dimension. Le logiciel HASTUS fabrique l'horaire de travail en trois phases :

1ère phase. Le logiciel partitionne les blocs en pièces de travail réalisables en solutionnant HASTUS-MACRO, une relaxation de la formulation HASTUS.

2e phase. Le logiciel fabrique une solution réalisable grâce à la résolution d'un problème de couplage.

3e phase. Le logiciel améliore localement la solution grâce à des procédures heuristiques.

La relaxation HASTUS-MACRO

Plusieurs modifications sont apportées à la formulation dans le but de rendre plus facile sa résolution :

La première modification à formulation HASTUS est l'élimination des variables y_{ij}. Cette élimination entraîne l'agrégation des contraintes (2.11) et l'obtention de la courbe du nombre N_t d'autobus en opération en tout moment t de la journée. Les contraintes (2.9) sont agrégées de façon à obtenir

$$\sum_{ijkh \,\in\, I(t)} x_{ijkh} \geq N_t \qquad t \in T. \qquad (2.14)$$

où T : ensemble des heures correspondant à un point de relève, un début ou une fin de bloc;

$I(t)$: l'ensemble des journées $(ijkh) \in I$ tel que $i < t < j$ ou $k < t < h$, $t \in T$;

N_t : le nombre d'autobus en opération durant la période débutant au temps $t \in T$.

$$N_t = \sum_{p \in L} \sum_{\substack{k < t \\ k \in T_p}} b_k^p$$

La seconde modification est la réduction de l'ensemble T en discrétisant le temps. Les seuls éléments admissibles de T sont les temps t=k*LGPÉRIODE. La division du temps en intervalles [t,t+LGPÉRIODE-1] réduit le nombre de journées x_{ijkh} nécessaires et le nombre de contraintes (2.14) nécessaires. Enfin, l'intégralité des variables x_{ijkh} a été relaxée.

Soit

T : les heures correspondant aux débuts d'intervalles;

I(t) : l'ensemble des journées $(ijkh)\in I$ tel que $i < t < j$ ou $k < t < h$, $t\in T$.

L(t) : l'ensemble des journées $(ijkh)\in I$ avec une pièce débutant en t (i=t ou k=t), $t\in T$;

N_t : le nombre d'autobus en opération durant la période débutant au temps t, $t\in T$;

Q_{pq} : le nombre de blocs du temps p au temps q, $(p,q\in T)$.

K_t : le nombre de blocs débutant au temps t, $t\in T$.

$$\text{Min} \sum_{ijkh\in I} c_{ijkh} x_{ijkh} \qquad (2.15)$$

sujet à

$$\sum_{ijkh\in I(t)} x_{ijkh} \geq N_t \qquad \text{pour tout } t\in T \qquad (2.16)$$

$$Dx \geq d \qquad (2.17)$$

$$\sum_{ijkh\in I(pq)} x_{ijkh} \geq Q_{pq} \qquad \text{pour tous les pq correspondant à un petit bloc} \qquad (2.18)$$

$$\sum_{ijkh\in L(t)} x_{ijkh} \geq K_t \qquad \text{pour tout } t\in T \qquad (2.19)$$

$$x_{ijkh} \geq 0 \qquad ijkh \in I. \qquad (2.20)$$

Les contraintes (2.16) assurent qu'à toutes les périodes de la journée le nombre de conducteurs est plus grand ou égal à la demande. Les contraintes (2.17) sont inchangées. Les contraintes (2.18) sont associées aux petits blocs. Les petits blocs sont ceux qu'il est impossible de diviser et

qui doivent former une pièce de travail. Ces contraintes garantissent que le nombre de journées ayant une pièce pq est au moins égale au nombre de petits blocs pq.

Les contraintes (2.19) exigent que le nombre de journées ayant une pièce débutant en t soit plus grand ou égal au nombre de blocs débutant en t. Les contraintes (2.18) et (2.20) n'étaient pas présentes dans la formulation HASTUS. Elles sont ajoutées à HASTUS-MACRO pour améliorer la réalisabilité de la solution obtenue.

La relaxation HASTUS-MACRO permet d'obtenir la forme des pièces composant la solution. Il nous faut maintenant obtenir les pièces légales partitionnant les blocs en solutionnant en nombre entier le problème de flot à coût quadratique suivant :

$$\text{Min} \sum_{ijkh \in I} \Big(\sum_{I(ij)} \bar{x}_{mnkh} - \sum_{p} y^{p}_{ij} \Big)^2$$

sujet à

$$\sum_{i \in T_p} y^{p}_{ik} - \sum_{j \in T_p} y^{p}_{kj} = b^{p}_{k} \quad \text{pour tout } k \in T_p, \text{ pour tout } p \in L$$

$$y^{p}_{ik} = \{0,1\} \quad \text{pour tout } ij \in I(ij), \text{ pour tout } p \in L$$

où $\bar{x}_{mnkh}$ est la solution optimale du problème HASTUS-MACRO.

$$y^{p}_{ij} = \begin{cases} 1 & \text{si la pièce ij est utilisée pour partionner le bloc p.} \\ 0 & \text{sinon.} \end{cases}$$

Ce problème est résolu par une procédure heuristique décrite dans [ROUS 85b].

Lorsque les blocs sont partitionnés en pièces légales, il est possible de résoudre le problème de couplage suivant pour assembler les pièces en journées de travail.

$$\text{Min} \sum_{I'} c_{ijkh} x_{ijkh}$$

$$\sum_{I'(ij)} x_{ijkh} = 1 \qquad (ij) \in V$$

$$x_{ijkh} = 0,1$$

où V : l'ensemble des pièces obtenues en partitionnant les blocs.

I' : l'ensemble des journées réalisables utilisant des pièces de V.

I'(ij) : l'ensemble des journées réalisables (mnkh)$\in$I' recouvrant la pièce ij.

Rousseau et al. [ROUS 85b] utilisent une formulation de flot pour résoudre le problème de couplage. La solution du problème de couplage peut violer les contraintes (2.10). Ce problème correspond à la troisième phase de Ball et al. La troisième phase a pour objectif d'effectuer des améliorations locales à la solution et d'obtenir une solution respectant les contraintes (2.10).

Les principales étapes de l'amélioration locale sont :

1^0 élimination des pièces vacantes par échange;

2^0 pour chaque bloc, le logiciel vérifie s'il est possible d'améliorer la solution en modifiant la partition du bloc en pièces;

3^0 si les pièces ont été modifiées, réoptimiser le problème de couplage.

D'autres méthodes d'obtention d'une solution réalisable, à partir de la relaxation HASTUS-MACRO ont été testées par Carraresi, Gallo et Rousseau [CARR 82]. Ces méthodes étaient basées sur la relaxation lagrangienne des contraintes de réseau (2.11).

Le logiciel HASTUS a été appliqué dans plusieurs villes et a permis des économies de l'ordre de 3% du coût dans des villes comme Montréal et Québec. Il permet de résoudre des problèmes comportant jusqu'à 350 journées de travail.

2.5 CONCLUSION

Au début des années 70, une première génération de méthodes informatisées de fabrication d'horaires de travail pour conducteurs d'autobus a été mise au point. Ces méthodes heuristiques reproduisaient les méthodes de modifications et d'échanges de pièces utilisées manuellement par les répartiteurs dans les sociétés de transport urbain.

La seconde génération de méthodes informatisées utilise la programmation mathématique pour fabriquer des horaires de travail. Les trois méthodes présentées dans ce chapitre ne permettent pas de garantir que l'horaire fabriqué est optimal. Dans le cas du modèle de recouvrement, le rejet a priori des journées de travail jugées non profitables, empêche de garantir l'optimalité car la solution optimale pourrait être constituée de beaucoup de journées de travail "profitables" et d'une "non-profitable". Dans le cas des deux autres méthodes, la séparation des phases de fabrication des pièces et d'assemblage de ces pièces en journées rend ces méthodes heuristiques.

Nous voulons dans cette thèse proposer une méthode dont la résolution complète permettrait de garantir l'optimalité de la solution obtenue. Cette méthode combinera le modèle de recouvrement et une méthode pour produire toutes les journées de travail permettant d'améliorer la solution.

CHAPITRE III

LA FABRICATION D'UN HORAIRE DE TRAVAIL POUR LES CONDUCTEURS D'AUTOBUS URBAINS PAR GÉNÉRATION DE COLONNES

Il existe un très grand nombre de journées de travail réalisables dans un problème d'horaire de travail. La fabrication d'un horaire avec une formulation de recouvrement demande d'énumérer toutes les journées de travail réalisables et de résoudre un problème de recouvrement où chaque variable représente une journée de travail. La résolution de ce problème devient impossible dès qu'il atteint une certaine taille. Toutefois, des problèmes de recouvrement ayant un grand nombre de variables ont été résolus par l'utilisation de la génération de colonnes. Les nouvelles variables (colonnes) sont générées par la résolution d'un sous-problème. Dans ce chapitre nous exposerons comment cette méthode peut être utilisée pour fabriquer des horaires de travail pour les conducteurs d'autobus urbains.

3.1 LA GÉNÉRATION DE COLONNES

La première utilisation explicite de la génération de colonnes est dûe aux travaux classiques de Gilmore et Gomory [GILM 61, GILM 63] sur le découpage des rouleaux de papier. Gilmore et Gomory minimisent le nombre de rouleaux de papier de largeur standard L à découper pour répondre à la demande N_i de rouleaux de largeur ℓ_i. Ce problème de recouvrement multiple se formule comme suit :

$$\text{Min} \sum_j x_j \tag{3.1}$$

$$\text{s.a.} \quad \sum_j a_{ij} x_j \geq N_i \quad \text{pour chaque largeur } \ell_i \quad i=1,\ldots,m \tag{3.2}$$

$$x_j \text{ entier} \tag{3.3}$$

où x_j : nombre de rouleaux de largeur L découpés selon le patron j.

a_{ij}: nombre de rouleaux de largeur ℓ_i dans le patron de découpe x_j (a_{ij} entier).

Dans un problème type, Gilmore et Gomory veulent découper des rouleaux de largeur standard L=200 pouces en rouleaux de 40 largeurs différentes variant de 20 à 80 pouces. Pour un tel problème, il y a entre 10 et 100 millions de colonnes réalisables. Pour chaque colonne (patron de découpe), la largeur totale des rouleaux obtenus doit être inférieure à la largeur standard d'un rouleau :

$$\sum_{i=1}^{m} \ell_i a_{ij} \leq L \qquad \text{pour tout } j=1,\ldots,n. \qquad (3.4)$$

Lors de la résolution par programmation linéaire du problème (3.1-3.2), nous restreignons le problème en ne considérant qu'un ensemble réduit de variables N'. Lors du calcul des coûts marginaux des variables et du choix de la variable d'entrée par l'algorithme du simplexe, deux cas sont possibles. Soit que nous n'avons pas atteint l'optimalité pour le problème réduit et nous effectuons le choix de la variable d'entrée dans l'ensemble N'. Sinon nous devons effectuer le choix de la variable d'entrée parmi toutes les variables possibles et ajouter la variable choisie à l'ensemble N', i.e. générer une colonne.

Soit u_i, la variable duale associée aux rouleaux de largeur ℓ_i (i=1,...,m); le coût marginal d'une variable x_j est égal au coût de la variable c_j moins la valeur marginale des rouleaux qui la composent :

$$\bar{c}_j = c_j - \sum_{i=1}^{m} a_{ij} u_i. \qquad (3.5)$$

Nous désirons obtenir la variable de coût marginal minimal respectant la contrainte (3.4). Ce problème est équivalent au problème de sac alpin suivant :

$$\text{Min } c_j - \sum_{i=1}^{m} u_i a_{ij} = - c_j - \text{Max} \sum_{i=1}^{m} u_i a_{ij} \qquad (3.6)$$

$$\text{s.a.} \quad \sum_{i=1}^{m} \ell_i a_{ij} < L \qquad (3.7)$$

$$a_{ij} \text{ entier} \qquad (3.8)$$

Si le coût marginal calculé est négatif, alors la nouvelle variable est la variable d'entrée et est ajoutée à l'ensemble N'. Sinon nous n'avons pas trouvé de variable d'entrée et la solution optimale du problème réduit est la solution optimale du problème (3.1,3.2). La solution entière du problème est obtenue par arrondissement, les auteurs ayant constaté que l'augmentation de la fonction objective était négligeable.

Crainic [CRAI 78, CRAI 86] a été le premier à utiliser la génération de colonnes pour la fabrication d'horaires de travail pour les équipages d'avions. L'objectif est de recouvrir tous les vols d'un équipage au moindre coût. L'auteur utilise une méthode d'énumération pour générer les colonnes de coût marginal négatif. La solution entière est obtenue en utilisant la méthode heuristique de Lemke, Salkin et Spielberg [LEMK 71].

Lavoie, Minoux et Odier [LAVO 85] fabriquent aussi des horaires de travail pour les équipages d'avions. Toutefois, ils produisent les colonnes de coût marginal négatif par un algorithme de plus court chemin dans une série de réseaux. Les coûts associés aux horaires d'un membre d'équipages ne sont pas linéaires et doivent être approchés lors du calcul des plus courts chemins. La construction d'une série de réseaux ayant chacun pour source (premier vol) un vol i spécifié permet la génération rapide des colonnes. Cette méthode a permis de résoudre des problèmes comportant jusqu'à 132 segments de vol.

Desrosiers, Soumis et Desrochers [DESR 84a] ont fabriqué des routes d'autobus scolaires en utilisant la génération de colonnes. L'objectif est

de minimiser les coûts reliés aux autobus (acquisitions d'autobus et engagement de conducteurs) et au kilométrage parcouru pour exécuter toutes les tâches. Les routes doivent recouvrir toutes les tâches

$$\text{Min} \sum c_r y_r$$

$$\text{s.a.} \quad \sum a_{ir} y_r \geq 1 \qquad \text{pour toutes les tâches } i$$

$$y_r \in \{0,1\}$$

où y_r : la route y_r est utilisée

$$a_{ir} = \begin{cases} 1 & \text{si la tâche } i \text{ fait partie de la route } r. \\ 0 & \text{sinon.} \end{cases}$$

Chaque tâche doit débuter dans une fenêtre de temps $[a_i, b_i]$ et une route d'autobus $(i_1, i_2, \ldots, i_k)$ est légale si toutes les heures de début des tâches T_i sont comprises dans $[a_i, b_i]$. Les colonnes de coût marginal négatif sont la solution d'un problème de plus court chemin avec fenêtres de temps. Ce problème est défini dans un réseau $G=(N,A)$ où les noeuds sont les tâches et où un arc (i,j) représente une transition possible entre les tâches i et j. Les noeuds p et q représentent respectivement le début et la fin d'une route.

$$\text{Min} \sum_{(i,j)\in A} (c_{ij} - \pi_i)\, x_{ij}$$

$$\text{s.a.} \quad \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = \begin{cases} +1 & i=p \\ -1 & i=q \\ 0 & i\in N,\ i\neq p,\ i\neq q. \end{cases}$$

$$x_{ij} > 0 \rightarrow T_i + d_{ij} \leq T_j \qquad (i,j) \in A$$

$$a_i \leq T_i \leq b_i \qquad i \in N$$

où les variables sont

T_i : heure de début d'une tâche.

x_{ij} : la tâche j est effectuée immédiatement après la tâche i.

et les données sont

π_i : valeur marginale de la tâche i.
c_{ij} : coût de la transition entre la tâche i et la tâche j.
d_{ij} : durée de la transition entre la tâche i et la tâche j.
$[a_i, b_i]$: fenêtre de temps associée au début de la tâche i.

Desrosiers, Pelletier et Soumis [DESR 83] ont mis au point un algorithme de programmation dynamique pour résoudre ce problème. Cet algorithme est une généralisation de l'algorithme de Ford-Bellman-Moore pour le problème de plus court chemin. La résolution du problème de recouvrement (3.1-3.3) est effectuée par programmation linéaire. La solution entière du problème de recouvrement est obtenue par l'utilisation d'une méthode de branchement mise au point par Carpaneto et Toth [CARP 80] pour le problème du commis-voyageur à coût asymétrique et adapté pour les routes d'autobus. Soit une route (i,j,k,l) dont la valeur est fractionnaire, i.e. $0 < y_r = x_{ij} = x_{kl} < 1$. Ce branchement sépare le problème en quatre sous-problèmes

1) $x_{ij} = x_{jk} = x_{kl} = 1$
2) $x_{ij} = x_{jk} = 1 \quad x_{kl} = 0$ la transition (k,l) est interdite
3) $x_{ij} = 1 \quad x_{jk} = 0$ la transition (j,k) est interdite
4) $x_{ij} = 0$ la transition (i,j) est interdite

Cette méthode a permis de résoudre des problèmes comportant jusqu'à 223 tâches.

Cullen, Jarvis et Ratliff [CULL 81] et Cullen [CULL 84] ont fabriqué des routes de camions avec contraintes de capacité en utilisant une méthode de génération de colonnes. Leur méthode est basée sur la notion de route primitive. Une route primitive est une forme géométrique (point, segment de droite ou ellipse) approchant la forme d'une route. Le coût d'ajout d'un point à une route j est fonction de la distance de ce point à la route primitive j. Les auteurs utilisent deux méthodes pour fabriquer des routes.

La fabrication d'une seule route s'effectue en solutionnant un problème de sac-alpin. La grandeur du sac-alpin représente la capacité du camion. La seconde méthode permet la fabrication d'une série de k routes recouvrant tous les points. La première étape de la méthode consiste à définir les routes primitives recouvrant bien les points à visiter et à définir les coûts d_{ij} d'ajout du point i à la route j pour les k routes. Les routes sont fabriquées en résolvant le problème d'affectation généralisée suivant:

$$\text{Min} \quad \sum_{i \in N} \sum_{j=1}^{k} d_{ij} x_{ij}$$

$$\text{sujet à} \quad \sum_{j=1}^{k} x_{ij} = 1 \qquad \text{pour tout } i \in N$$

$$\sum_{i \in N} r_i x_{ij} \leq R \qquad j=1,\ldots,k$$

$$x_{ij} = 0,1 \qquad \text{pour tout } i \in N,\ j=1,\ldots k$$

où x_{ij} : le point i fait partie de la route j;
r_i : demande du point i;
R : capacité du camion.

Les premières contraintes exigent que chaque point fasse partie d'une route et la seconde série exige que la capacité des camions soit respectée.

La solution globale du problème est obtenue à l'aide d'un problème de partitionnement. De plus, l'interaction homme-machine permet de modifier les routes et de suggérer l'emplacement et la forme de routes primitives. Cette méthode a permis la résolution de plusieurs problèmes ayant jusqu'à 200 points. De plus, un problème de 1 000 points et 5 dépôts a été solutionné.

Desrosiers, Dumas et Soumis [DESR 84b] et Dumas [DUMA 85] ont utilisé une méthode semblable à celle de Desrosiers et al. [DESR 84a] pour

fabriquer des horaires pour le transport urbain adapté aux handicapés et des routes pour des camions. Dans ces deux types de problèmes, une tâche consiste à prendre en charge un handicapé (ou des marchandises) d'un lieu de départ et de le conduire à sa destination. Durant le trajet jusqu'à la destination du passager, le véhicule peut prendre en charge de nouveaux passagers ou déposer à leur destination d'anciens passagers, si ces activités n'entraînent pas de dépassement de la capacité du véhicule ou de non-respect de la fenêtre de temps sur l'arrivée à la destination d'un des passagers. Les horaires doivent aussi respecter des contraintes de priorité (un passager doit être pris en charge avant d'être déposé) et de couplage (un passager doit être pris en charge et déposé par un même véhicule).

Les deux problèmes demandent la résolution d'un problème de recouvrement avec des contraintes additionnelles et utilisent la méthode de branchement de Carpaneto-Toth [CARP 80] adaptée par Desrosiers et al. [DESR 84a] pour obtenir une solution en nombres entiers. Les nouvelles colonnes sont générées par des problèmes de plus court chemin avec des contraintes supplémentaires. Toutefois, même si les contraintes des deux problèmes sont les mêmes, les données de problèmes types de chacun des problèmes présentent suffisamment de différences pour être traités par des algorithmes différents.

Dans les problèmes types de fabrication de tournées pour les handicapés, les contraintes de capacité du véhicule sont négligeables et les contraintes de fenêtres de temps sur l'heure d'arrivée au lieu de départ et à la destination sont très contraignantes. Desrosiers et al. [DESR 84b] réduisent la complexité des problèmes en créant heuristiquement des groupes de clients. Chaque groupe est servi séparément. Pour chaque groupe, nous connaissons l'horaire d'un véhicule permettant de prendre en charge et de déposer tous les passagers tout en respectant les contraintes de capacité,

de priorité et de couplage. Pour chaque groupe, il est possible de calculer la fenêtre de temps telle que l'horaire du véhicule est réalisable. La création de colonnes composées de groupes permet la génération d'horaires réalisables et d'éliminer les contraintes de capacité, de priorité et de couplage. L'algorithme de Desrosiers, Pelletier et Soumis [DESR 83] pour le calcul du plus court chemin avec contraintes d'horaires est utilisé pour générer les colonnes. Des problèmes comportant jusqu'à 200 clients divisés en 65 groupes ont été résolus par cette méthode.

Dans les problèmes types de fabrication de tournées de camions, les contraintes de fenêtres de temps sont négligeables tandis que les contraintes de capacité sont très actives. Dumas [DUMA 85] génère les colonnes grâce à un algorithme de programmation dynamique permettant le calcul du plus court chemin avec contraintes de capacité, de fenêtre de temps, de priorité et de couplage. Cette méthode permet la fabrication de tournées déservant jusqu'à 55 clients.

La génération de colonnes a été utilisée pour résoudre divers problèmes. Toutefois, ces problèmes ont tous en commun d'être formulés par le recouvrement de tâches à exécuter au moyen de variables réalisables (patron de découpe, journée de travail, tournée ou horaire d'un véhicule). De plus, dans tous les cas nous connaissons une méthode pour calculer la variable de coût marginal le plus négatif. La génération de colonnes a l'avantage d'être une méthode primale et de permettre à l'usager le choix entre l'obtention d'une solution optimale ou d'une solution réalisable de coût acceptable.

3.2 LES HORAIRES DE TRAVAIL DE CONDUCTEURS D'AUTOBUS URBAINS

Présentement l'utilisation de la formulation de recouvrement permet de fabriquer des horaires de travail pour de petits problèmes comportant moins de 100 journées de travail. Tous les logiciels existant effectuent un

choix préliminaire des journées de travail pour ne garder qu'un nombre raisonnable de variables. Ce choix réduit de variables ne permet pas de garantir l'optimalité de la solution obtenue. L'utilisation de la génération de colonnes permet de garantir l'optimalité. De plus, les résultats numériques de Desrosiers et al. [DESR 84a, DESR 84b] et Dumas [DUMA 85] montrent que le nombre de colonnes générées est environ le quintuple du nombre de tâches dans le problème de recouvrement, tandis que les expériences de Heurgon [HEUR 75] et Marsten et Shepardson [MARS 81a] montrent que le nombre de variables considérées varie de 10 à 50 fois le nombre de tâches du problème de recouvrement. Dans ce cas, la génération de colonnes pourrait aussi permettre des économies de mémoire lors de la résolution du simplexe.

Le problème-maître

Lors de la fabrication d'horaires de travail pour les conducteurs d'autobus urbains le problème-maître est un problème de recouvrement avec des contraintes additionnelles.

Le problème consiste à fabriquer l'horaire de coût minimum

$$\text{Min} \sum_{j \in J} c_j x_j \tag{3.9}$$

sujet à des contraintes de recouvrement

$$\sum_{j \in J} a_{ij} x_j \geq 1 \qquad \text{pour tout } i \in I \tag{3.10}$$

et des contraintes d'intégrité :

$$x_j = 0,1 \qquad \text{pour tout } j \in J \tag{3.11}$$

De plus, quelques contraintes additionnelles peuvent être imposées. Ces contraintes étant généralement imposées sous forme d'un pourcentage, nous imposons une contrainte dont le rôle est de compter les journées de

travail à l'exception des pièces vacantes

$$\sum_{j \in JBC} x_j - N = 0 \qquad (3.12)$$

où JBC : ensemble des indices des Journées Brisées ou Continues.

N : compteur du nombre de journées de travail dans la solution.

Les autres contraintes additionnelles sont du type énoncé au chapitre 1 de la description des paramètres de HASTUS :

a) les pourcentages minimum et maximum du nombre de journées continues est souvent contraint

$$\sum_{j \in JC} x_j - MIN1P*N \geq 0 \qquad (3.13)$$

$$\sum_{j \in JC} x_j - MAX1P*N \leq 0 \qquad (3.14)$$

où JC : ensemble des indices des journées continues;

b) le nombre de pièces vacantes est souvent limité

$$\sum_{j \in PV} x_j \leq NB\text{-}VACANT \qquad (3.15)$$

où PV : ensemble des indices de pièces vacantes;

c) une ou plusieurs contraintes peuvent limiter le pourcentage du nombre de journées de travail ayant A1 ou plus (ou moins) d'amplitude à un minimum (um maximum) de P2.

$$\sum_{AMPLj \geq (\leq) A1} x_j - P2*N \geq (\leq) 0 \qquad (3.16)$$

où AMPLj : amplitude de la journée de travail j.

L'exposé de ces quelques contraintes complète la formulation de recouvrement utilisée pour modéliser les horaires de travail. Nous verrons dans les prochaines sections comment générer les colonnes et résoudre le problème-maître.

3.3 LA MODÉLISATION DE LA GÉNÉRATION D'UNE JOURNÉE DE TRAVAIL

Dans trois implantations récentes de la méthode de génération de colonnes [DESR 84a, DESR 84b, DUMA 85], les nouvelles colonnes étaient produites grâce à un algorithme de plus court chemin avec des contraintes supplémentaires. Nous voulons produire les nouvelles journées de travail d'une façon similaire.

L'examen des paramètres HASTUS, dont une partie est exposée au chapitre 1, nous montre que l'admissibilité et le coût d'une journée de travail sont souvent définis en fonction de quatre valeurs :

- NBPCE, le nombre de pièces de travail dans une journée;
- AMPL, l'amplitude de la journée;
- HREPAY, le nombre d'heures payées pour la journée;
- HRETRA, le nombre d'heures travaillées durant la journée.

Il existe toutefois des conventions collectives où l'admissibilité et le coût d'une journée de travail sont définis d'une façon légèrement différente. Nous verrons dans le chapitre sur les résultats numériques un problème où le temps travaillé depuis le début de la "demi-journée" (i.e. depuis le début de la journée ou depuis la fin de la pause) remplace le temps travaillé depuis le début de la journée.

Dans notre problème de plus court chemin, nous voulons définir les arcs comme des parties de journées (pièce ou pause) et définir des contraintes supplémentaires sur les valeurs (NBPCE, AMPL, HREPAY, HRETRA). Ceci permettra la définition dans la prochaine section du réseau et des contraintes à l'aide des paramètres HASTUS. Avant de poursuivre cette définition, voyons quelques exemples de paramètres HASTUS. Le paramètre PIECE-LG-MIN contraint la durée minimale d'une pièce et permet d'éliminer les arcs représentant des pièces d'une durée inférieure. Le paramètre GARANTI-PIECE définit la durée payée minimale et le coût minimal sur un arc représentant une pièce. Le paramètre AMPLITUDE-MAXIMUM contraint l'amplitu-

de d'une journée et permet d'imposer une contrainte supplémentaire sur AMPL. Ces exemples permettent d'illustrer comment les paramètres HASTUS définissent notre exemplaire du problème de plus court chemin avec contraintes supplémentaires.

De façon générale, le problème de plus court chemin avec contraintes supplémentaires (PPCCCS) est défini sur un réseau G=(N,A) où N est l'ensemble des noeuds incluant une source p et un puits q et A est l'ensemble des arcs. Le problème de plus court chemin constitue la première partie du PPCCCS, et est défini par le coût $c_{ij} \in R$ associé à chaque arc $(i,j) \in A$ et par les contraintes de conservation de flot :

$$\text{Min } cx = \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{3.17}$$

sujet à

$$x \in X(G) = \begin{cases} \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = \begin{cases} 1, & i=p \\ 0, & i\in N,\ i\neq p, \\ -1, & i=q \end{cases} \\ x_{ij} \geq 0 \qquad \text{pour tout } (i,j)\in A. \end{cases} \tag{3.18}$$

De plus, des contraintes supplémentaires sont imposées sur L ressources. Dans notre exemplaire, ces ressources sont le nombre de pièces de travail, l'amplitude, les heures payées et les heures travaillées. Pour chaque arc $(i,j) \in A$, nous connaissons $d_{ij}^{\ell} \in N$ la quantité de la ressource ℓ utilisée par l'arc. Dans notre exemplaire, nous avons

$d_{ij}^{1} = \begin{cases} 0 & \text{si l'arc } (i,j) \text{ est une pièce ou un arc vers le puits.} \\ 1 & \text{sinon.} \end{cases}$

d_{ij}^{2} = durée de la pièce ou pause représentée par l'arc (i,j).

d_{ij}^{3} = heures payées de la pièce ou pause représentée par l'arc (i,j).

d_{ij}^{4} = heures travaillées de la pièce ou pause représentée par l'arc (i,j).

Soit T_j^ℓ, la quantité de la ressource ℓ utilisée en utilisant un chemin de la source jusqu'au noeud j; le PPCCCS consiste à trouver le chemin élémentaire de coût minimum entre la source et le puits tel que T_j^ℓ respecte un intervalle admissible $[a_j^\ell, b_j^\ell]$ pour tous les noeuds du chemin et toutes les ressources. Dans notre cas, les valeurs a_j^ℓ et b_j^ℓ seront déterminées par les valeurs des divers paramètres HASTUS. Il est possible d'atteindre un noeud avant le début de l'intervalle admissible d'un noeud en le débutant au début de cet intervalle en gaspillant la ressource. Par exemple, le minimum d'heures payées pour une journée, i.e. pour atteindre le puits, est GARANTI-JOURNEE et si à l'arrivée au puits le nombre d'heures payées est inférieur, il suffit de payer les heures manquantes pour obtenir une journée réalisable.

L'ajout des contraintes suivantes aux contraintes (3.17, 3.18) complète de PPCCCS :

$$x_{ij} > 0 \rightarrow T_i^\ell + d_{ij}^\ell \leq T_j^\ell \qquad (i,j)\in A,\ \ell=1,\ldots,L$$

$$a_i^\ell \leq T_i^\ell \leq b_i^\ell \qquad i\in N,\ \ell=1,\ldots,L$$

où x_{ij} représente le flot sur l'arc (i,j), et T_i^ℓ, la quantité de la $\ell^{ième}$ ressource utilisée pour atteindre le noeud i.

Le réseau

Nous définissons les arcs de notre réseau des pièces ou des pauses, alors que chaque tâche i est représentée par deux noeuds. Le premier noeud (2i-1) représente le début de la tâche i et le second (2i) représente la fin de la tâche. Un arc (2i-1,2j) représentant une pièce joint le début d'une tâche i et la fin d'une tâche j du même bloc si la convention collective autorise cette pièce. Un arc (2i, 2j-1) représentant une pause joint la fin de la tâche i et le début de la tâche j si le déplacement entre les deux points de relève est possible et si la convention collective autorise cette pause. Débuter une journée avec la tâche j est représenté par un arc

(p,2j-1) et terminer une journée par la tâche i est représenté par un arc (2i,q).

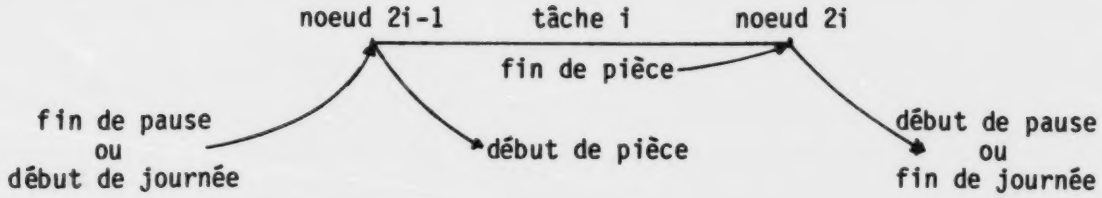


Figure 3.1 : Exemple des différents types d'arcs

Nous connaissons pour chaque bloc l'heure de début, de fin et de passage aux points de relève. Nous pouvons donc déterminer DEBUT(j) et FIN(j), les heures de début et de fin de la tâche j. Les paramètres HASTUS permettent de définir l'existence, le coût et les ressources utilisées par un arc pour chacune des catégories d'arcs :

a) arc (p, 2i-1) représentant un début de journée par la tâche i

coût : $c_{p,2i-1}$ = COUT-HORAIRE*(PREPAR-SORTIE + DUREE (GARAGE, NOEUD 2i-1)

nombre de pièces : $d^1_{p,2i-1} = 1$

amplitude : $d^2_{p,2i-1}$ = PREPAR-SORTIE + DUREE (GARAGE, NOEUD 2i-1)

heures payées : $d^3_{p,2i-1}$ = PREPAR-SORTIE + DUREE (GARAGE, NOEUD 2i-1)

heures travaillées : $d^4_{p,2i-1}$ = PREPAR-SORTIE + DUREE (GARAGE, NOEUD 2i-1)

Il est à noter que dans certaines conventions collectives ce temps de préparation fait partie de la première pièce.

b) arc (2i,q) représentant une fin de journée

coût : $c_{2i,\,q}$ = COUT-HORAIRE*(PREPAR-RENTREE + DUREE (GARAGE, NOEUD 2i)

nombre de pièces : $d^1_{2i,q} = 0$

amplitude : $d^2_{2i,q}$ = PREPAR-RENTREE + DUREE (GARAGE, NOEUD 2i)

heures payées : $d^3_{2i,q}$ = PREPAR-RENTREE + DUREE (GARAGE, NOEUD 2i)

heures travaillées : $d^4_{2i,q}$ = PREPAR-RENTREE + DUREE (GARAGE, NOEUD 2i)

Dans certaines conventions collectives, le temps de retour au garage fait partie de la dernière pièce de la journée.

c) arc (2i, 2j-1) représentant une pause

Un arc de pause n'est possible que si le déplacement entre le lieu de la fin de la tâche i et celui du début de la tâche j est possible. De plus, la durée de la pause est limitée.

dur_{ij} = DEBUT(j) - FIN(i) - DUREE (NOEUD 2i, NOEUD 2j-1)

L'arc existe si dur_{ij} est compris entre PAUSE-MIN-BRIS et PAUSE-MAX-BRIS si nous cherchons des journées brisées ou entre PAUSE-MIN-CONTINUE et PAUSE-MAX-CONTINUE si nous cherchons des journées continues.

coût : $c_{2i,2j-1}$ = COUT-HORAIRE*(DUREE NOEUD 2i,NOEUD 2j-1)) + TAUX-PAUSE-$[^{\text{BRIS ou}}_{\text{CONTINUE}}]$*$dur_{ij}$

nombre de pièces : $d^1_{2i,2j-1}$ = 1

amplitude : $d^2_{2i,2j-1}$ = DEBUT(j) - FIN(i)

heures payées : $d^3_{2i,2j-1}$ = DUREE(NOEUD 2i,NOEUD 2j-1) + TAUX-PAUSE $[^{\text{BRIS ou}}_{\text{CONTINUE}}]$*$dur_{ij}$

heures travaillées : $d^4_{2i,2j-1}$ = DUREE(NOEUD 2i,NOEUD 2j-1)

Dans certaines conventions collectives, le déplacement vers le dépôt est compris dans la pièce précédant la pause.

d) arc (2i-1, 2j) représentant une pièce de travail

Un arc de pièce recouvre plusieurs tâches et ces tâches doivent faire partie du même bloc. Le coût d'une pièce est composé du coût des heures payées moins les valeurs duales π_k des tâches recouvertes. De plus, la durée de la pièce FIN(j)-DEBUT(i) doit être comprise entre PIECE-LG-MIN et PIECE-LG-MAX.

Soit L_{ij}, l'ensemble des tâches recouvertes par la pièce représentée par l'arc (2i-1, 2j), alors

coût : $c_{2i-1,2j} = \text{COUT-HORAIRE}*(\text{FIN}(j)-\text{DEBUT}(i)) - \sum_{k \in L_{ij}} \pi_k$

nombre de pièces : $d^1_{2i-1,2j} = 0$

amplitude : $d^2_{2i-1,2j} = \text{FIN}(j)-\text{DEBUT}(i)$

heures payées : $d^3_{2i-1,2j} = \text{MAX}\ \{\text{GARANTI-PIECE}, \text{FIN}(j)-\text{DEBUT}(i)\}$

heures travaillées : $d^4_{2i-1,2j} = \text{FIN}(j)-\text{DEBUT}(i)$

Dans certaines conventions collectives, les déplacements vers le dépôt et les allocations sont comprises dans la pièce de travail.

Les contraintes supplémentaires

Les contraintes supplémentaires limitent les valeurs que les ressources (nombre de pièces, amplitude, heures payées ou heures travaillées) peuvent prendre. Les paramètres HASTUS définissent les valeurs des intervalles admissibles $[a_i^\ell, b_i^\ell]$. De façon générale, pour chacune des ressources et chacun des noeuds, les intervalles sont

a) pour le nombre de pièce $[a_i^1, b_i^1]$ = [0,PIECE-NB-MAX] pour tous les noeuds.

b) pour l'amplitude $[a_i^2, b_i^2]$ = [0,AMPLITUDE-MAXIMUM] pour tous les noeuds.

c) pour les heures payées $[a_i^3, b_i^3]$ = $[0, \infty]$ pour tous les noeuds.

Dans plusieurs conventions collectives, le nombre d'heures payées est égal au nombre d'heures travaillées et cette ressource n'est pas considérée lors du calcul du plus court chemin.

d) pour les heures travaillées $[a_i^4, b_i^4]$ = [0,JOURNEE-LG-MAX] pour tous les noeuds.

De plus, certaines contraintes sont imposées sur un seul noeud. La garantie sur le nombre d'heures payées redéfinit la fenêtre de réalisabilité des heures payées du puits comme

$$[a_q^3, b_q^3] = [\text{GARANTI-JOURNEE}, \infty].$$

D'autres paramètres HASTUS non-exposés au chapitre 1 peuvent définir de façon similaire les intervalles admissibles des noeuds ou à un noeud en particulier.

Les autres coûts sur les arcs

Dans certains cas, le coût sur un arc (i,j) ne dépend pas seulement de l'arc mais aussi de la quantité de ressources déjà utilisées pour atteindre le noeud i. L'exemple le plus connu est la prime (pénalité) payée en cas de temps supplémentaire. Si un conducteur travaille plus de GARANTI-JOURNEE, i.e. utilise plus de GARANTI-JOURNEE de la ressource heures travaillées, le coût de chaque heure de travail est majoré de SUPPL-TAUX*COUT-HORAIRE. De façon semblable un boni est associé aux journées ayant une grande amplitude.

1) Augmentation du taux horaire (ex. : temps supplémentaire)

Si $T_i^4 + d_i^4 >$ GARANTI-JOURNEE, le taux horaire est augmenté de SUPPL-TAUX pour la portion dépassant GARANTI-HORAIRE et le coût de l'arc devient

$$c_{ij} := c_{ij} + \text{SUPPL-TAUX*COUT-HORAIRE*}\min\{d_{ij}^4, T_i^4, d_{ij}^4 - \text{GARANTI-JOURNEE}\}$$

2) boni (ex. : boni pour grande amplitude, contraintes supplémentaires du problème maître).

Nous voulons ajouter un boni au coût des journées présentant certaines caractéristiques (p.e amplitude dépassant AMPL). Si $T_i^2 <$ AMPL et $T_i^2 + d_{ij}^2 >$ AMPL alors nous ajoutons le boni au coût de l'arc (i,j) :

$$c_{ij} := c_{ij} + \text{AMPLITUDE-BONI}$$

Nous pouvons traiter comme un boni les valeurs duales associées aux contraintes additionnelles du problème maître. Ces contraintes portent généralement sur l'appartenance à un ensemble de journées de travail (grande amplitude, journées continues, etc.). Si une journée devient membre de l'ensemble associé à une contrainte, il faut ajouter la valeur duale au coût de l'arc.

Contraintes de qualification des arcs

Dans certains cas, l'existence d'un arc (i,j) dépend de la quantité de ressources utilisées pour parvenir au noeud i. Par exemple, lors de la fabrication d'une journée continue, un arc (i,j) représentant une fin de journée n'existe que si, pour la ressource "heures travaillées",

$$\text{CONTINUE-LG-MIN} \leq T_i^4 + d_{ij}^4 \leq \text{CONTINUE-LG-MAX}.$$

Cette condition de qualification sur l'arc (i,j) peut être utilisée pour modéliser l'interdiction du gaspillage de la ressource "heures travaillées".

La définition du réseau, des contraintes supplémentaires et des pénalités n'est pas complète puisque tous les paramètres HASTUS n'ont pas été exposés. Toutefois, les modèles exposés permettent de comprendre comment ces autres paramètres sont traités et quelle est la modélisation d'une journée de travail.

Les pièces vacantes

Les pièces vacantes sont une forme spéciale de journées de travail, que nous préférons ne pas modéliser comme un plus court chemin avec contraintes supplémentaires. Toutefois, les pièces vacantes étant constituées de tâches consécutives provenant d'un même bloc, il existe peu de pièces vacantes légales et il est aisé de les énumérer toutes et d'évaluer leur coût marginal. Soit L_{ij}, l'ensemble des tâches recouvertes par une pièce

vacante ayant comme première tâche, la tâche i et comme dernière, la tâche j, alors le coût marginal de cette pièce vacante est

$$\text{coût} = \text{COUT-HORAIRE}*\text{TAUX-VACANT}*\ \text{MAX}\{\text{FIN}(j)-\text{DEBUT}(j),\text{GARANTI-VACANT}\}-\sum_{k \in L_{ij}} \pi_k .$$

3.4 LA RÉSOLUTION DU PROBLÈME-MAÎTRE

Le problème-maître (3.9-3.16) tel qu'exposé à la section 3.2 est un problème de recouvrement avec contraintes additionnelles. Desrosiers et al. [DESR 84a, DESR 84b] et Dumas [DUMA 85] ont solutionné ce type de problème en relaxant les contraintes d'intégralité. La solution du problème relaxé (3.9-3.10,3.12-3.16) utilise un algorithme de programmation linéaire. Une solution en nombres entiers est obtenue en utilisant une méthode de séparation et d'évaluation progressive.

La figure 3.2 illustre l'algorithme de résolution du problème-maître. Le problème relaxé (3.9-3.10,3.12-3.16) est résolu par le logiciel XMP [MARS 81b] et les colonnes sont générées par un algorithme trouvant le plus court chemin avec contraintes supplémentaires. Dans la figure 3.2, cette partie de l'algorithme est contenue dans le carré central. Le reste de la figure concerne l'initialisation du problème et la méthode de branchement utilisée pour l'obtention d'une solution en nombres entiers.

Initialisation

Il existe une solution primale "réalisable" de coût élevé permettant l'initialisation du problème relaxé. Il s'agit d'utiliser uniquement des pièces vacantes pour recouvrir les tâches. Cette solution est réalisable (sauf la contrainte 3.15 qui pourrait être violée) et permet de définir des variables duales, de débuter la génération de colonnes et d'initialiser une borne supérieure $\bar{Z}$ sur le coût de l'horaire des conducteurs d'autobus. Lors d'une étape ultérieure, l'utilisation d'une pénalité sur les pièces vacantes permettra au besoin le respect de la contrainte 3.15.

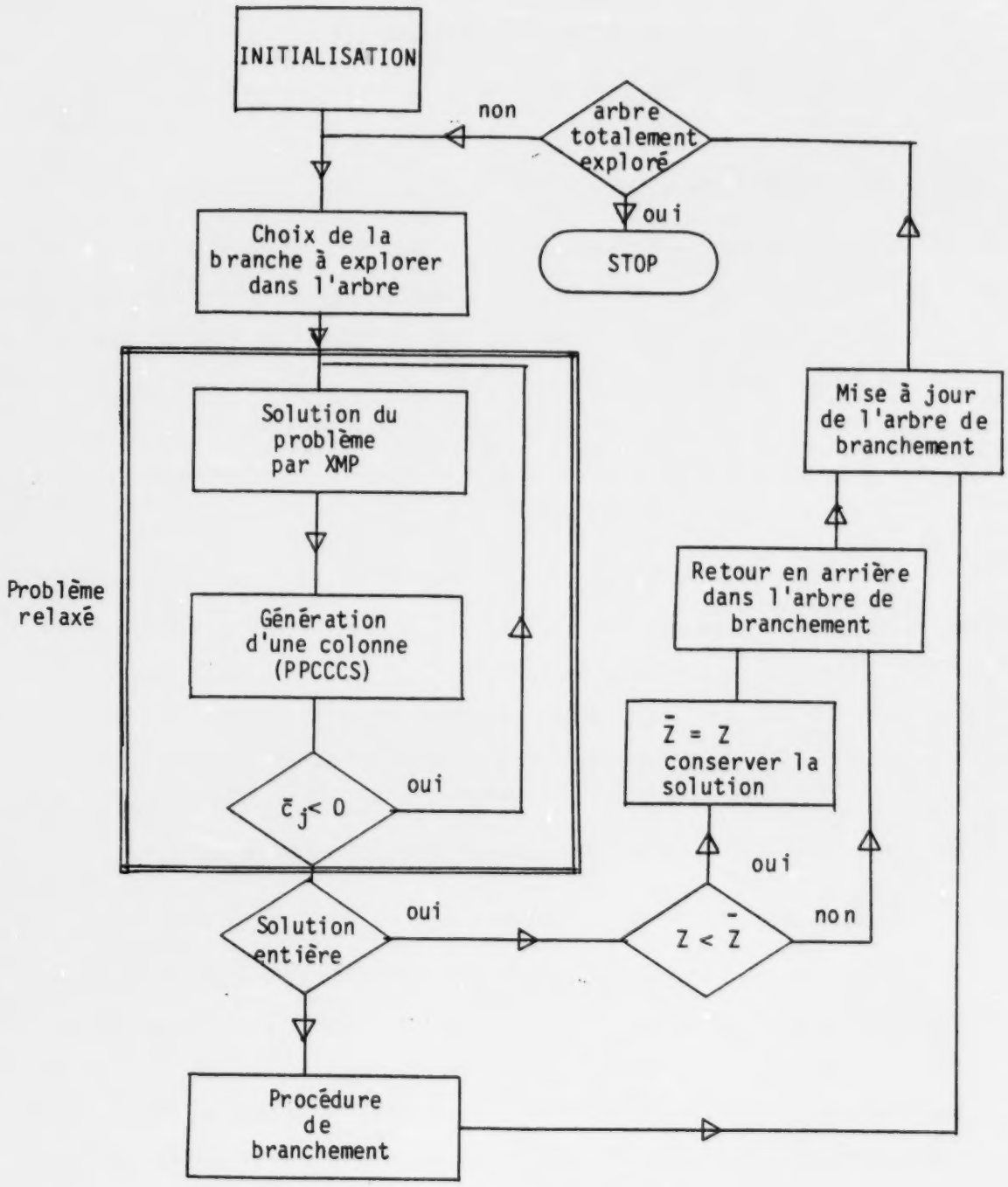


Figure 3.2 : Algorithme de résolution du problème maître

Il est préférable toutefois d'initialiser la solution avec des journées de travail réalisables. Le coût de la solution initiale obtenue est inférieur et les valeurs des variables duales auraient une meilleure répartition qu'avec une solution initiale composée uniquement de pièces vacantes.

Perturbation du problème

Les problèmes de recouvrement sont des problèmes très dégénérés car chaque variable recouvre exactement plusieurs contraintes. Pour éviter le cyclage souvent associé aux problèmes très dégénérés, nous perturbons légèrement le problème-maître lors de sa première résolution. Le problème-maître devient :

$$\text{Min } c.x$$

s.a.

$$Ax \geq 1+\varepsilon_1$$

$$Dx \leq d+\varepsilon_2$$

$$x \geq 0.$$

où ε_1 et ε_2 sont des vecteurs de petites valeurs aléatoires.

La perturbation du problème-maître permet de garantir une amélioration de la fonction objective à chaque itération.

Élimination de variables

Balas [BALA 80a] a défini des coupes pour la résolution des problèmes de recouvrement. La forme la plus simple de ces coupes est la fixation à zéro d'une variable, i.e. son élimination du problème. Soit le problème

$$\text{Min } \{cx \mid Ax \geq b,\ x \geq 0,\ x_j \text{ entier},\ j \in N_1 \subset N\} \qquad (P).$$

IBEST est une borne supérieure sur la valeur de (P). Le vecteur $u>0$ représente les valeurs duales de (P) et le vecteur $s = c-uA \geq 0$ représente

les valeurs marginales des variables de (P). Balas montre que l'on peut fixer à zéro les variables de $Q_0 = \{j \in N_1 \mid s_j > \text{IBEST-ub}\}$ car toute solution réalisable x de (P) telle que $cx < \text{IBEST}$ satisfait $x_j = 0$, pour tout $j \in Q_0$. Cette élimination des variables permet une réduction de la taille des problèmes de programmation linéaire à résoudre.

Modification des contraintes additionnelles

Si les contraintes additionnelles (3.12-3.16) telles que présentées modélisent bien les contraintes de la convention collective, leur forme actuelle n'est pas toujours propice à l'obtention d'une solution entière. Par exemple, soit une contrainte imposant un minimum de 50% de journées continues :

$$\sum_{j \in JC} x_j - \frac{N}{2} \geq 0$$

Il existe des cas où cette inégalité est contraignante dans la solution continue et non contraignante dans la solution entière. Ces cas se produisent si N, le nombre total de journées de travail, est impair. Alors, il existe une différence d'au moins 0,5 entre le nombre de journées continues et N/2 dans toute solution entière. Pour obtenir une inégalité valide à la fois dans les solutions continues et entières, il faut modifier la contrainte de la façon suivante :

$$\sum_{j \in JC} x_j \geq \left\lceil \frac{N}{2} \right\rceil$$

De façon générale, si la convention collective impose un miminum de MINP % sur le nombre de journées de travail appartenant à l'ensemble S, la contrainte modifiée (renforcée) sera

$$\sum_{j \in S} x_j \geq \left\lceil \frac{\text{MINP*N}}{100} \right\rceil .$$

De façon semblable, si la convention collective impose un maximum de MAXP % sur le nombre de journées de travail appartenant à l'ensemble S, la

contrainte modifiée (renforcée) sera

$$\sum_{j \in S} x_j \leq \left\lfloor \frac{MAXP \cdot N}{100} \right\rfloor .$$

Cette transformation évite d'avoir à explorer l'écart entre la contrainte originale et la contrainte renforcée par branchement. Toutefois, dans un contexte de programmation linéaire, cette transformation n'est possible que si nous connaissons la valeur de N (ou un estimé assez précis). Cet estimé peut être obtenu de deux façons :

- Souvent l'exploitant pose une borne supérieure sur le nombre de journées de travail nécessaires. Cette borne supérieure est généralement très précise.
- La résolution du problème avant modification permet de calculer N, un bon estimé du nombre de journées de travail dans la solution entière.

Après avoir procédé à la modification des contraintes et obtenu une nouvelle solution continue, il ne reste qu'à obtenir une solution entière par branchement.

La méthode de branchement

La dernière étape de la fabrication d'horaires de travail pour les conducteurs d'autobus urbains est l'obtention d'une solution en nombres entiers pour le problème maître. Deux méthodes de branchement ont déjà été utilisées sur de tels problèmes. Desrosiers, Soumis et Desrochers [DESR 84a] ont adapté le branchement de Carpaneto-Toth [CARP 80] originellement développé pour le problème de commis-voyageur. Ce branchement fixe une série de tâches. Ryan et Forster [RYAN 81], pour un problème d'horaires de travail, ont mis au point un branchement sur les paires de tâches.

Dans notre problème maître, chaque journée de travail est représentée par une variable et est composée d'une suite de tâches. Nous allons construire un petit exemple constitué de 5 blocs divisés en 13 tâches. Cet exemple permettra d'illustrer les méthodes de branchement.

		Début	Fin
Bloc 1	Tâche 1	08:00	09:30
Bloc 1	Tâche 2	09:30	11:00
Bloc 1	Tâche 3	11:00	12:30
Bloc 1	Tâche 4	12:30	14:00
Bloc 1	Tâche 5	14:00	15:30
Bloc 2	Tâche 6	07:00	09:00
Bloc 2	Tâche 7	09:00	10:30
Bloc 3	Tâche 8	07:00	08:30
Bloc 3	Tâche 9	08:30	10:30
Bloc 4	Tâche 10	14:30	16:30
Bloc 4	Tâche 11	16:30	18:00
Bloc 5	Tâche 12	14:30	16:00
Bloc 5	Tâche 13	16:00	18:00

Cet exemple ayant pour seul but d'exposer les méthodes de branchement, nous énumérerons toutes les journées réalisables selon une convention collective. Nous examinerons plus tard les effets des méthodes de branchement sur la génération de colonnes. Selon la convention, chaque journée aura au moins 8 heures payées à un taux horaire de 10,00 $. De plus, au delà de 10 heures d'amplitude un boni de 5,00 $ est versé au conducteur. Les pièces vacantes seront à temps et demi. Seules les journées de travail suivantes ont été jugées réalisables. La solution réalisable obtenue est :

$x_1 = 0,0$	$c_1 = 80$	$A_1 = \{1,2,10\}$
$x_2 = 0,5$	$c_2 = 80$	$A_2 = \{1,2,10,11\}$
$x_3 = 0,5$	$c_3 = 80$	$A_3 = \{1,2,12,13\}$
$x_4 = 0,0$	$c_4 = 80$	$A_4 = \{1,2,4,5\}$
$x_5 = 0,0$	$c_5 = 80$	$A_5 = \{1,2,3,10,11\}$
$x_6 = 0,0$	$c_6 = 80$	$A_6 = \{1,2,3,12,13\}$
$x_7 = 0,0$	$c_7 = 80$	$A_7 = \{2,10,11\}$
$x_8 = 0,0$	$c_8 = 80$	$A_8 = \{2,12,13\}$
$x_9 = 0,5$	$c_9 = 80$	$A_9 = \{3,4,5,6,7\}$
$x_{10} = 0,5$	$c_{10} = 80$	$A_{10} = \{3,4,5,8,9\}$
$x_{11} = 0,0$	$c_{11} = 80$	$A_{11} = \{4,5,6,7\}$

$x_{12} = 0,0$	$c_{12} = 80$	$A_{12} = \{4,5,8,9\}$
$x_{13} = 0,0$	$c_{13} = 85$	$A_{13} = \{6,7,10,11\}$
$x_{14} = 0,5$	$c_{14} = 85$	$A_{14} = \{6,7,12,13\}$
$x_{15} = 0,5$	$c_{15} = 85$	$A_{15} = \{8,9,10,11\}$
$x_{16} = 0,0$	$c_{16} = 85$	$A_{16} = \{8,9,12,13\}$

De plus, définissons les pièces vacantes suivantes :

$x_{17} = 0,0$	$c_{17} = 22,50$	$A_{17} = \{1\}$
$x_{18} = 0,0$	$c_{18} = 30,00$	$A_{18} = \{11\}$

où x_j : valeur de la variable j dans la solution.

c_j : coût de la journée de travail j.

A_j : tâches i recouvertes par la journée de travail j.

Le branchement de Ryan et Forster sélectionne une paire de tâches (t_1 et t_2) exécutées dans au moins une même journée (variable) telle que

$$0 < \sum_{j \in J(t_1,t_2)} x_j < 1$$

où $J(t_1,t_2)$: ensemble des variables j où la tâche t_1 et la tâche t_2 sont exécutées.

Dans toute solution entière, les variables de $J(t_1,t_2)$ doivent ou recouvrir totalement ces tâches ou ne pas les recouvrir. Nous pouvons séparer le problème en deux sous-problèmes selon

branche-0 $$\sum_{j \in J(t_1,t_2)} x_j \leq 0$$

branche-1 $$\sum_{j \in J(t_1,t_2)} x_j \geq 1.$$

Ryan et Forster montrent que l'ajout de ces contraintes élimine du problème les sous-matrices interdites dans une matrice parfaite, i.e. élimine les sous-matrices permettant l'apparition d'une solution fractionnaire du problème. La branche-0 impose que les deux tâches ne soient pas recouvertes par la même journée de travail tandis que la branche-1 impose l'exécution commune des deux tâches par une journée de travail. Dans notre exemple, les couples de tâches (1,10), (1,11), (1,12), (1,13), (2,10), (2,11), (2,12), (2,13), (3,6), (3,7), (3,8), (3,9), (4,6), (4,7), (4,8), (4,9), (5,6), (5,8), (7,12), (9,10), etc. étaient éligibles pour un branchement. Toutefois, plusieurs de ces couples produisent les mêmes branches. Un branchement sur le couple (2,10) aurait produit les branches :

$$\text{branche-0} \qquad x_1 + x_2 + x_5 + x_7 = 0$$
$$\text{branche-1} \qquad x_1 + x_2 + x_5 + x_7 = 1$$

Toutefois, cette méthode de branchement s'adapte assez mal avec le sous-problème de fabrication de journées de travail car il est difficile d'imposer l'exécution disjointe ou commune de deux tâches quelconques dans un problème de plus court chemin.

L'adaptation de Desrosiers, Soumis et Desrochers de la méthode de branchement de Carpaneto-Toth sélectionne une variable (journée de travail) fractionnaire. Si cette journée recouvre les tâches, la méthode sépare le problème original en k sous-problèmes selon les règles suivantes :

sous-problème 1 $\quad \sum_{j \in S(t_1,\ldots,t_k)} x_j \geq 1$

sous-problème 2 $\quad \sum_{j \in S(t_1,\ldots,t_{k-1},\bar{t}_k)} x_j \geq 1 \quad \text{et} \quad \sum_{j \in T(t_{k-1},t_k)} x_j \leq 0$

sous-problème 3 $\quad \sum_{j \in S(t_1,\ldots,t_{k-2},\bar{t}_{k-1})} x_j \geq 1 \text{ et } \sum_{j \in T(t_{k-2},t_{k-1})} x_j \leq 0$

sous-problème k $\quad \sum_{j \in S(t_1,\bar{t}_2)} x_j \geq 1 \quad \text{et} \quad \sum_{j \in T(t_1,t_2)} x_j \leq 0$

où $t_1,\ldots,t_k$: la suite des tâches recouvertes par la journée j.

$S(t_1,\ldots,t_k)$: l'ensemble des variables j exécutant les tâches $t_1,\ldots,t_k$ dans cet ordre.

$S(t_1,\ldots,t_i,\bar{t}_{i+1})$: l'ensemble des variables j exécutant les tâches t_1 à t_i dans cet ordre et n'exécutant pas la tâche i+1 ou ne l'exécutant pas immédiatement après la tâche i.

$T(t_i,t_j)$: ensemble des journées recouvrant consécutivement i et j.

Dans notre exemple, les 6 variables positives peuvent être utilisées pour définir un branchement. L'utilisation de la variable x_2 nous donne les 4 sous-problèmes suivants :

sous-problème 1 $x_2 > 1$

sous-problème 2 $x_1 > 1$ et $x_2 + x_5 + x_7 + x_{13} + x_{15} < 0$

sous-problème 3 $x_3 + x_4 + x_5 + x_6 > 1$ et $x_1 + x_2 + x_7 < 0$

sous-problème 4 $x_{17} > 1$ et $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 < 0$

Cette méthode de branchement est bien adaptée au sous-problème de fabrication de journées de travail car il suffit de modifier le réseau pour imposer l'exécution ou la non-exécution consécutive de deux tâches.

Dans notre exemple, l'évaluation des 4 sous-problèmes produit les solutions suivantes :

sous-problème 1 : coût = 245
$x_2 = 1,0$
$x_9 = x_{10} = x_{14} = x_{16} = 0,5$

sous-problème 2 : coût = 275
$x_1 = x_{18} = 1,0$
$x_9 = x_{10} = x_{14} = x_{16} = 0,5$

sous-problème 3 : coût = 245
$x_3 = 1,0$
$x_9 = x_{10} = x_{13} = x_{15} = 0,5$

sous-problème 4 : coût = 267,50

$x_{17} = 1,0$

$x_7 = x_8 = x_9 = x_{10} = x_{14} = x_{15} = 0,5$

Il est à noter que la qualité des sous-problèmes n'est pas la même. Les sous-problèmes 1 et 3 réduisent le nombre de variables à valeur fractionnaire sans augmenter le coût de la solution. Les sous-problèmes 2 et 4 produisent une augmentation du coût des solutions. Dans le cas du sous-problème 4, le nombre de variables à valeur fractionnaire n'a pas diminué.

La branche 1 est productive car elle a forcé l'utilisation de la journée de travail x_2. La branche 3 est productive car elle force la non-utilisation de journée de travail (x_1, x_2, x_7) dont la somme des valeurs était 0,5 dans le problème original. Les branches 2 et 4 sont inutiles car elles forcent la non-utilisation de journées de travail $(x_2, x_5, x_7, x_{13}, x_{15})$ et $(x_1, x_2, x_3, x_4, x_5, x_6)$ dont la somme des valeurs était 1,0 dans le problème original. Les branches 2 et 4 vont donc à l'encontre de la solution continue. Cette méthode de branchement génère des sous-problèmes inutiles chaque fois que dans la solution, deux tâches i et j sont toujours exécutées consécutivement, i.e.

$$\sum_{j \in T(t_1, t_2)} x_j = 1.$$

Les seuls sous-problèmes utiles sont donc ceux où

$$0 < \sum_{j \in T(t_1, t_2)} x_j < 1.$$

Nous pouvons noter la similarité entre cette condition et celle du branchement de Ryan et Forster. La seule différence est au niveau de la définition de $J(t_1, t_2)$ et $T(t_1, t_2)$ et il existe des cas où ces ensembles peuvent être identiques. Il s'agit de toutes les paires de tâches telles que si elles sont exécutées par une même journée, elles doivent y être exécutées consécutivement car il est impossible d'exécuter une autre tâche

entre ces deux tâches. Par exemple, deux tâches consécutives dans un bloc exécutées par la même journée doivent être exécutées consécutivement. Dans notre exemple, les ensembles $J(t_1,t_2)$ et $T(t_1,t_2)$ sont identiques pour les couples (2,10), (2,12), (3,7), (3,9), etc.

Plutôt que d'utiliser le branchement de Carpaneto-Toth, nous utiliserons un branchement semblable défini par une paire de tâches de façon à ne produire que des sous-problèmes utiles. Pour produire des sous-problèmes utiles, la condition

$$0 < \sum_{j \in T(t_1,t_2)} x_j < 1$$

doit être vérifiée par la paire de tâches et les sous-problèmes suivants seront produits

$$\text{sous-problème 1} \qquad \sum_{j \in T(t_1,t_2)} x_j > 1$$

$$\text{sous-problème 0} \qquad \sum_{j \in T(t_1,t_2)} x_j < 0$$

Pour rendre la méthode de branchement efficace, il faut choisir les paires de tâches de façon à ce que $T(t_1,t_2)$ soit identique à l'ensemble $J(t_1,t_2)$ du branchement de Ryan et Forster.

Pour atteindre ce but, nous choisirons d'abord de brancher sur les paires de tâches t_1, t_2 consécutives dans un bloc. Si ces deux tâches sont utilisées dans un journée de travail, alors elle doivent faire partie de la même pièce de travail. Sinon elles font partie de journées de travail différentes et elles doivent faire partie de deux pièces de travail et il y aura changement de conducteur au point de relève entre les 2 tâches. Un branchement sur cette paire de tâches permettra de décider entre ces deux éventualités. Dans la branche-1, nous forçons t_1 et t_2 à faire partie de la même pièce de travail. Dans la branche-0, nous forçons ces tâches à faire partie de pièces de travail différentes. Ce choix de tâches permet de déterminer les pièces de travail de la solution.

Si toutes les pièces de travail sont déterminées, i.e. il n'existe aucune paire de tâches (t_1,t_2) consécutives dans un bloc tel que la somme des valeurs des variables $T(t_1,t_2)$ soit fractionnaire, nous devons choisir une paire (t_1,t_2) correpondant à une pause, i.e. la tâche t_1 termine une pièce et la tâche t_2 débute une autre pièce de travail. Puisqu'il peut être possible d'insérer l'exécution d'une autre tâche t_3 entre t_1 et t_2, les ensembles $J(t_1,t_2)$ et $T(t_1,t_2)$ peuvent être différents et le branchement moins fort.

Méthode de branchement et génération de colonnes

Il est facile de modifier le réseau du problème de fabrication des journées de travail pour que les journées fabriquées soient acceptables dans le sous-problème actuel.

A) si les tâches i et j appartenant au même bloc doivent être exécutées consécutivement, alors tous les arcs représentant une pièce exécutant i mais pas j ou inversement sont éliminés;

B) si les tâches i et j appartenant au même bloc ne doivent pas être exécutés consécutivement, alors tous les arcs représentant une pièce exécutant consécutivement i et j sont éliminés;

C) si les tâches i et j n'appartenant pas au même bloc doivent être exécutées consécutivement, alors tous les arcs représentant une pause entre i et une autre tâche que j (et inversement) sont éliminés.

D) si les tâches i et j n'appartenant pas au même bloc ne doivent pas être exécutées consécutivement, alors l'arc représentant une pause joignant i et j est éliminé.

3.5 CONCLUSION

Dans ce chapitre, nous avons exposé une méthode utilisant la génération de colonnes pour la fabrication d'horaires de travail pour les conducteurs d'autobus urbains. Cette méthode diffère des méthodes précédentes de génération de colonnes [DESR 84a, DESR 84b, DUMA 85] en plusieurs points.

Lors de la résolution du problème-maître, nous utilisons une nouvelle méthode de branchement qui devrait permettre des branches équilibrées et un faible nombre de sous-problèmes comme celle de Ryan et Forster. Cette nouvelle méthode demeure adaptée à la génération de colonnes comme celle de Desrosiers, Soumis, et Desrochers.

La modélisation de la fabrication des journées de travail a amené la formulation d'une problème de plus court chemin avec plusieurs contraintes supplémentaires. Les algorithmes existants [DESR 83] permettent de résoudre des problèmes de plus court chemin avec une contrainte supplémentaire avec environ 300 noeuds. La fabrication d'horaires de travail de taille raisonnable demandera le développement dans les prochains chapitres d'algorithmes d'optimisation et de réoptimisation pour des problèmes avec plusieurs contraintes supplémentaires.

CHAPITRE IV

UN ALGORITHME DE MARQUAGE GÉNÉRALISÉ POUR LE PROBLÈME DE PLUS COURT CHEMIN AVEC CONTRAINTES D'HORAIRES

Inconnues il y a quelques années, les contraintes d'horaires sont maintenant utilisées pour modéliser les heures d'ouvertures des clients, les heures préférées de livraison, etc. dans plusieurs problèmes de fabrication de routes et d'horaires. Le problème de plus court chemin avec contraintes d'horaires (PPCCCH) a été utilisé par Desrosiers, Soumis et Desrochers [DESR 84a] pour fabriquer des routes d'autobus scolaires par génération de colonnes. Le PPCCCH est un problème NP-difficile et nous présentons dans ce chapitre le premier algorithme pseudo-polynomial pour le résoudre.

4.1 LE PROBLÈME

Le problème de plus court chemin avec contraintes d'horaires consiste à trouver le chemin de coût minimum entre deux noeuds dans un réseau en arrivant à chaque noeud i visité à l'intérieur de son invervalle de temps $[a_i,b_i]$. Soit le réseau $G=(N,A)$ où l'ensemble de noeuds N est constitué de tâches, d'une source p et d'une destination q. À chaque arc (i,j) est associé un coût c_{ij} négatif ou positif comprenant le coût inter-tâches (i,j) ainsi que le coût de la tâche i et une durée d_{ij} positive comprenant la durée inter-tâches (i,j) et la durée de la tâche i. Étant donné les fenêtres de temps $[a_i,b_j]$ pour chacune des tâches, tous les arcs $(i,j)\in A$ permettent, en exécutant la tâche i au plus tôt, de débuter la tâche j avant la fin de sa fenêtre de temps, i.e. tous les arcs (i,j) respectent la condition :

$$a_i + d_{ij} < b_j \qquad (4.1)$$

En utilisant cette notation, nous pouvons formuler le problème du plus court chemin élémentaire avec contraintes d'horaires (PPCCCH) de la façon suivante :

$$\text{Minimiser} \quad c.x = \sum_{(i,j)\in A} c_{ij}\, x_{ij} \tag{4.2}$$

sujet à

$$x \in X(G) = \begin{cases} \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = \begin{cases} +1 & i=p \\ 0 & i\in N,\ i\neq p,\ i\neq q \\ -1 & i=q \end{cases} & (4.3) \\ x_{ij} > 0 \qquad \text{pour tout } (i,j)\in A & (4.4) \end{cases}$$

$$x_{ij} > 0 \Rightarrow t_i + d_{ij} \leq t_j \quad (i,j)\in A \tag{4.5}$$

$$a_i \leq t_i \leq b_i \qquad i\in N \tag{4.6}$$

où x_{ij} représente le flot sur l'arc (i,j) et t_i, le temps de début de la tâche i.

Ce problème apparaît comme sous-problème lors de la fabrication de routes d'autobus scolaires. Nous voulons déterminer le nombre et la composition des routes permettant de réaliser toutes les tâches de façon à minimiser le coût total. Desrosiers, Soumis et Desrochers [DESR 84a] utilisent la méthode de génération de colonnes pour construire des routes recouvrant de façon optimale l'ensemble des tâches à effectuer. À chaque itération de la méthode de génération de colonnes, nous cherchons à améliorer la solution actuelle en insérant dans la base la route de coût marginal minimum comprenant un sous-ensemble des tâches respectant les contraintes d'horaires. Cette route est obtenue en solutionnant le problème du plus court chemin avec contraintes d'horaires (PPCCCH).

4.2 REVUE DE LA LITTÉRATURE

Le problème du plus court chemin avec contraintes d'horaires n'est pas traité fréquemment dans la littérature mais quelques cas particuliers ont déjà été étudiés.

Le plus court chemin avec une contrainte supplémentaire

Il s'agit du cas particulier le plus souvent traité [JOKS 66, YUCA 73, MINO 75, HAND 80, ANEJ 83, JAFF 84, BEAS 84].

Deux familles d'algorithmes utilisant l'une la programmation dynamique et l'autre la relaxation lagrangienne sont proposées pour résoudre ce problème.

La programmation dynamique a été proposée pour la première fois par Joksch [JOKS 66] en 1966. Sa méthode résoud les équations de récurrence des solutions efficaces par une généralisation de l'algorithme de Ford-Bellman pour le calcul du chemin le plus court. Habenicht [HABE 81] a proposé une méthode semblable. Son implantation permet de résoudre des problèmes de très faible densité (1%) de 50 à 500 noeuds. Yucaoglu [YUCA 73] propose une méthode basée sur une généralisation de l'algorithme de Dijkstra. Minoux [MINO 75] a proposé une méthode comparable. Aneja, Aggarwal et Nair [ANEJ 83] proposent une méthode semblable pour résoudre une version du problème avec plusieurs contraintes supplémentaires. Cette méthode vise le calcul d'un sous-ensemble pertinent de solutions efficaces à chaque noeud. La méthode utilise plusieurs bornes pour limiter le nombre de solutions examinées. Jaffe [JAFF 84] propose une méthode basée sur l'algorithme de Ford- Bellman pour calculer la solution à tous les états possibles plutôt que de ne considérer que les solutions efficaces. Il étudie de façon détaillée l'effet sur la précision de la solution de ne calculer qu'une partie des états possibles.

La relaxation lagrangienne que propose Minoux [MINO 75] permet d'obtenir, grâce à la résolution d'un problème de plus court chemin avec coûts modifiés par l'addition d'un multiplicateur associé à la contrainte supplémentaire, une solution réalisable et une borne inférieure sur la valeur de la solution optimale. Handler et Zang [HAND 80] proposent une

méthode semblable en y ajoutant une seconde phase permettant l'obtention de la solution optimale. Cette seconde phase utilise le calcul du $K^{ième}$ plus court chemin avec coûts modifiés pour fermer le saut de dualité. Cette méthode permet de résoudre des problèmes de faible densité (1% à 10%) comportant de 50 à 500 noeuds.

Beasley et Christofides [BEAS 84] ont proposé une méthode semblable pour le cas multidimensionnel. Ils explorent le saut de dualité grâce à une procédure de branchement. Cette méthode permet de résoudre des problèmes de faible densité (2% à 10%) comportant de 100 à 500 noeuds et ayant 10 dimensions.

Ribeiro [RIBE 83] a traité une extension : le problème de plus court chemin avec double inégalité. Ribeiro propose plusieurs méthodes duales pour résoudre ce problème. Ces méthodes basées sur la relaxation lagrangienne et la méthode des sous-gradients pour l'ajustement des multiplicateurs de Lagrange permet d'obtenir des solutions réalisables et d'en prouver l'optimalité pour des problèmes de moyenne et forte densité (25% à 100%) de 50 à 100 noeuds.

Le problème bicritère

Le problème bicritère consiste à rechercher tous les chemins efficaces entre la source et la destination en minimisant deux critères indépendants. Ce problème diffère du PPCCCH car ses solutions n'ont pas à respecter de contraintes d'horaires.

Vincke [VINC 74] a proposé pour le cas où les deux coûts sont positifs, une généralisation de l'algorithme de Ford-Bellman pour trouver les solutions efficaces du problème bicritère. Pour le même cas, Hansen [HANS 80] propose une généralisation de l'algorithme de Dijkstra permettant de produire en temps pseudo-polynomial l'ensemble complet des solutions

efficaces. Toujours pour le même cas, Climaco et Martins [CLIM 82] proposent une méthode basée sur le calcul des K plus courts chemins entre la source et la destination de façon à énumérer toutes les solutions efficaces.

Martins [MART 84] étudie le problème multicritère et propose une généralisation de l'algorithme de Dijkstra. Il montre que soit c^i, i=1,...,K, les vecteurs de coûts de problème multicritère, il suffit qu'il existe un des $c^i > 0$ pour tout arc et qu'il n'y aie pas de cycle négatif pour que l'algorithme converge. Warburton [WARB 86] propose pour le même problème un schème d'approximation polynomial.

Le problème du plus court chemin avec contraintes d'horaires

Le PPCCCH a été formulé par Desrosiers, Pelletier et Soumis [DESR 83] comme un sous-problème pour une méthode de fabrication des routes [DESR 84a]. Les auteurs y prouvent que la solution optimale du PPCCCH (4.2)-(4.6) est entière et formulent le principe d'optimalité suivant : soit un chemin X_{pj} allant de la source p au noeud j, si ce chemin est efficace et si l'arc (i,j) est le dernier arc de X_{pj}, alors, le sous-chemin X_{pi} est un chemin efficace. Ce principe d'optimalité est utilisé pour mettre au point un algorithme appelé PCCCH basé sur la programmation dynamique. Cet algorithme est une généralisation de l'algorithme de Ford- Bellman.

L'algorithme de Ford-Bellman fonctionne par correction d'étiquette pour trouver une solution. Un noeud peut être traité plusieurs fois (réinséré) avant de trouver la solution finale. Desrosiers et al. utilisent l'implantation de D'Esopo-Pape, nommée L-2deque par Gallo et Pallottino [GALL 86] pour choisir l'ordre de traitement des noeuds. Cette implantation permet généralement de traiter efficacement la liste, toutefois, la complexité en pire cas est de $O(2^n)$ réinsertions. Cette méthode est sensible à l'ordre initial des données et il suffit de trier les noeuds en

ordre croissant des débuts de fenêtre de temps pour accélérer l'algorithme. Cet algorithme permet de résoudre des problèmes de petite et moyenne densité (15%-30%) ayant jusqu'à 300 noeuds.

Presque tous les algorithmes proposés pour le problème de plus court chemin avec contraintes supplémentaires [ANEJ 83, HAND 80, HABE 81, JAFF 84, JOKS 66, RIBE 83] et le problème de plus court chemin bicritère [HANS 80, VINC 74] exigent des coûts et des durées positifs. Seul Martins [MART 84] propose un algorithme pour le problème de plus court chemin multicritère convergent s'il n'y a pas de cycle négatif. Le PPCCCH ne pose pas de restrictions sur les coûts car les contraintes d'horaires empêchent la formation de cycles infinis.

L'algorithme PPCCCH de Desrosiers et al. [DESR 83] peut être amélioré en utilisant un autre ordre de traitement, soit la méthode L-2queue [GALL 86], dans la généralisation de l'algorithme de Ford-Bellman. L'algorithme résultant est pseudo-polynomial et a une complexité d'ordre $O(D^2)$ où $D = \sum_{i=1}^{n} (b_i - a_i + 1)$ est le nombre maximum de chemins efficaces. Nous proposerons plutôt un ordre de traitement des chemins efficaces et définirons des seaux généralisés qui permettrons d'un algorithme de type Dijkstra ayant une complexité d'ordre $O(D^2)$.

4.3 QUELQUES CARACTÉRISTIQUES DE PPCCCH

À chaque chemin X_{pj} de l'origine p à un noeud j satisfaisant les équations (4.3)-(4.6), est associée une étiquette (temps, coût) correspondant respectivement à l'heure d'arrivée au noeud j et au coût du chemin X_{pj}. Ces étiquettes seront représentées par (T_j^k, C_j^k) dénotant les caractéristiques du $k^{ième}$ chemin de p à i, les indices k ou i pourront être absents si le contexte n'est pas ambigu.

Le calcul de cette étiquette est effectué itérativement le long du chemin $X_{pj} = (i(0), i(1), i(2), \ldots, i(L))$, où $i(0) = p$ et $i(L) = j$, de la

façon suivante :

$$T_{i(0)} = 0$$

$$C_{i(0)} = 0$$

$$T_{i(\ell)} = \max\{a_{i(\ell)}, T_{i(\ell-1)} + d_{i(\ell-1)i(\ell)}\} \qquad \ell=1,\ldots,L$$

$$C_{i(\ell)} = C_{i(\ell-1)} + c_{i(\ell-1)i(\ell)} \qquad \ell=1,\ldots,L$$

Définition 4.1. Soit X^1 et X^2 deux chemins différents de p à j et leurs étiquettes associées (T^1,C^1) et (T^2,C^2), X^1 domine X^2 ou $(T^1,C^1) < (T^2,C^2)$ si et seulement si $(T^2,C^2)-(T^1,C^1) > (0,0)$ et $(T^1,C^1) \neq (T^2,C^2)$.

La relation de dominance n'est pas une relation d'ordre total et elle ne permet pas d'ordonner tous les chemins. La seule chose que la relation de dominance nous permet de conclure sur un chemin X_{pj} non dominé (efficace) est que X_{pj} est le plus court chemin arrivant au noeud j à l'heure T_j ou avant. Ceci implique la possibilité de plusieurs chemins efficaces pour un noeud. La figure 4.1 illustre la relation entre différentes étiquettes (temps, coût) associées à des chemins efficaces (X^1 à X^4) et à d'autres dominés (X^5 à X^8).

Cette relation permet de définir le coût du chemin réalisable de coût minimal comme une fonction de temps d'arrivée. La figure 4.1 permet d'illustrer cette fonction coût versus temps pour un noeud.

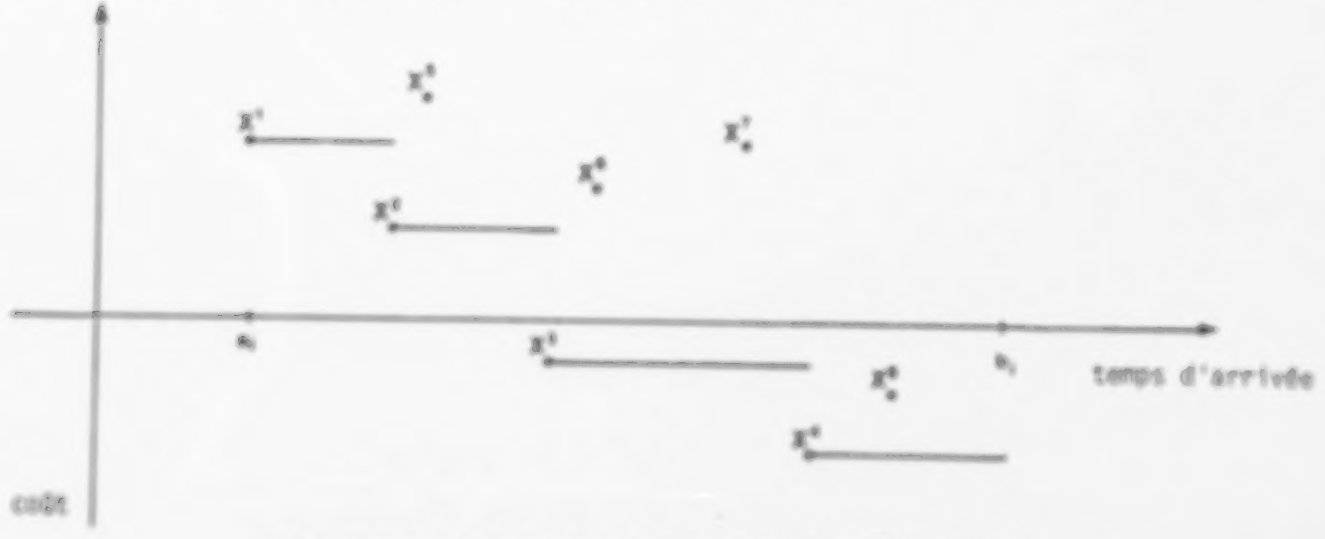


Figure 4.1 : Relation de dominance entre étiquettes associées à différents chemins.

Desrosiers et al. utilisent une adaptation de l'algorithme de Ford-Bellman. Puisque la solution associée à un noeud est composée d'un ensemble de solutions efficaces, la principale adaptation se situe à l'étape d'exploration des successeurs du noeud traité. L'algorithme adapté trouve les successeurs de i par le processus suivant :

Soit $Q_i = \{(T_i^k, C_i^k)\}, k=1,\ldots,K(i)$ l'ensemble des étiquettes efficaces au noeud i, en ajoutant l'arc $(i,j) \in A$ aux K(i) chemins associés à Q_i, nous obtenons de nouveaux chemins pour le noeud j. Nous pouvons calculer l'ensemble des étiquettes associées à ces chemins par l'application :

$$F_{ij}(Q_i) = \bigcup_{k=1}^{K(i)} \begin{cases} \{(\max[a_j, T_i^k + d_{ij}],\ C_i^k + C_{ij})\} \\ \qquad \text{si } T_i^k + d_{ij} < b_j \\ \\ \emptyset \quad \text{sinon.} \end{cases}$$

L'ensemble $F_{ij}(Q_i)$ est fusionné avec l'ensemble Q_j et ce nouvel ensemble Q_j est réduit de façon à ne conserver que les étiquettes efficaces du noeud j. Si Q_j a été modifiée par cette étape, il faut ajouter le noeud j à la liste des noeuds à traiter.

4.4 UN NOUVEL ORDRE DE TRAITEMENT DES ÉTIQUETTES

Dans les algorithmes de plus courts chemins sans contraintes, les notions de noeud et d'étiquettes sont jumelées, l'ordre de traitement est donc défini à la fois sur les noeuds et sur les étiquettes. Dans le cas du plus court chemin avec contraintes supplémentaires, ces deux notions sont distinctes car un ensemble d'étiquettes est associé à chaque noeud. Lors de la mise au point d'algorithmes pour ces problèmes, généralement, l'ordre de traitement porte sur les noeuds et l'algorithme résultant est de type Ford-Bellman [JOKS 66, VIMK 74, DESR 83, JAFF 84]; pour le cas parti-

culier où les coûts et les durées sont positifs, l'algorithme résultant est une adaptation simple de type Dijkstra [YUCA 73, MINO 75, ANEJ 83, HANS 80]. La seule exception est l'algorithme de type Dijkstra pour le problème multicritère de Martins [MART 84] utilisant un ordre de traitement défini sur les étiquettes. Toutefois, sa seconde condition de qualification sur les arcs, aucun cycle négatif sur un des vecteurs coûts, est difficile à vérifier et n'est pas satisfaite par beaucoup d'exemplaires du PPCCCH.

L'existence d'un algorithme de type Dijkstra pour les PPCCCH dépend de l'existence d'un ordre de traitement des étiquettes nous assurant, avec une condition de qualification sur les arcs, qu'il est impossible d'améliorer une étiquette déjà traitée.

Définition 4.2. Soit a, b $\in R^2$, a est lexicographiquement plus petit que b($a \overset{L}{<} b$) si et seulement si $a \neq b$ et que le premier élément non-nul de b-a est positif.

Définition 4.3. Un arc (i,j) est lexicographiquement positif si et seulement si $(0,0) \overset{L}{<} (d_{ij},c_{ij})$.

L'ordre lexicographique est un ordre total dans R^2 compatible avec l'ordre partiel de dominance, i.e. que les trois propositions suivantes sont vraies :

1) pour tout a, b $\in R^2$, si $a < b$, alors $a \overset{L}{<} b$.
2) pour tout a, b $\in R^2$, si $a = b$, alors $a \overset{L}{=} b$.
3) pour tout a, b $\in R^2$, si $a > b$, alors $a \overset{L}{>} b$.

Théorème 4.1. Pour tout a, b $\in R^2$, si $a \overset{L}{<} b$, alors non ($a > b$) i.e. a ne peut dominer b.

Preuve. $a \overset{L}{<} b$ implique que $b \neq a$ et que le premier élément non-nul de b-a est positif, il existe donc une composante i où $a_i < b_i$, ce qui contredit $a > b$. □

Théorème 4.2. Soit P, l'ensemble des étiquettes déjà traitées et T, l'ensemble des étiquettes non traitées. Si 1) tous les arcs $(i,j) \in A$ sont lexicographiquement positifs, et 2) les étiquettes de T sont traitées dans l'ordre lexicographique croissant, alors pour tout $a \in P$, pour tout $b \in T$, $a \overset{L}{<} b$.

Lemme 4.1. Lors du traitement d'une étiquette, ses successeurs sont lexicographiquement supérieurs à cette étiquette.

Preuve du lemme

$$(T_i, C_i) + (d_{ij}, c_{ij}) = (T_j, C_j)$$

$$(T_i, C_i) \overset{L}{<} (T_j, C_j) \qquad (d_{ij}, c_{ij}) \overset{L}{>} 0, \text{ pour tout } (i,j) \in A.$$

Preuve du théorème. (par induction sur le nombre d'éléments de P)

1. $|P| = 0$ cas trivial.
2. Supposons le théorème vrai pour $|P| = k$, vérifions pour $|P^*| = k+1$.

Soit $(T_f, C_f) = \underset{(T_j^k, C_j^k) \in T}{\text{minlex}} \{(T_j^k, C_j^k)\}$ étiquette à traiter.

Le nouvel ensemble T* est l'union des anciens étéments de $T - \{(T_f, C_f)\}$ et des successeurs de (T_f, C_f), donc par le lemme 4.1 et par la définition de (T_f, C_f), pour tout $b \in T^*$, $(T_f, C_f) \overset{L}{<} b$.

Par hypothèse d'induction, nous avons pour tout $a \in P$, $a \overset{L}{<} (T_f, C_f)$.

Le nouvel ensemble P* étant l'union de P et de $\{(T_f, C_f)\}$, nous pouvons déduire que

pour tout $a \in P^*$, pour tout $b \in T^*$, $a \overset{L}{<} b$. □

Corollaire

Soit P, l'ensemble des étiquettes déjà traitées et
T, l'ensemble des étiquettes non traitées.

Si 1) tous les arcs $(i,j)\in A$ sont lexicographiquement positifs
2) les étiquettes de T sont traitées dans l'ordre lexicographique croissant

Alors le traitement de l'étiquette minimale de T ne peut pas améliorer une étiquette de P.

Preuve. Soit $b = \underset{(T_i^k,C_i^k)\in T}{\text{minlex}} \{(T_i^k,C_i^k)\}$ l'étiquette à traiter et

$S = \{c \mid c \text{ est un successeur réalisable de } b\}$.

Le lemme 1 implique

$$b \overset{L}{<} c \quad \text{pour tout } c\in S$$

Le théorème 2 implique que

$$a \overset{L}{<} b \quad \text{pour tout } a\in P$$

Donc $a \overset{L}{<} c$ pour tout $a\in P$, pour tout $c\in S$.

Le théorème 4.1 montre que c ne peut dominer a, donc que c ne peut améliorer a. □

Si la condition de qualification sur les arcs permet de garantir que l'algorithme termine en un temps fini, elle n'est toutefois pas suffisante pour garantir que le chemin trouvé sera élémentaire.

Lors de la génération de colonnes, des chemins non-élémentaires, i.e. des chemins ou au moins un noeud est visité plus d'une fois, sont souvent produits dans les premières itérations. Ces chemins non-élémentaires apparaissent s'il existe des déséquilibres dans les valeurs marginales des noeuds qui créent des cycles de coût négatif. Ces coûts disparaissant lorsque les valeurs marginales s'approchent de leurs valeurs finales. La présence d'un chemin non-élémentaire dans la solution finale du problème-maître est très improbable et le problème des cycles de coût négatif peut être ignoré.

4.5 L'ALGORITHME DE DIJKSTRA GÉNÉRALISÉ

Pour utiliser les résultats de la section précédente pour la mise au point d'un algorithme, il suffit de vérifier que pour le PPCCCH les coûts sur les arcs sont lexicographiquement positifs. Nous avons ordonné le couple de coûts dans l'ordre (durée, coût) car dans les problèmes avec contrainte d'horaire les durées d_{ij} (durées de la tâche i plus durée inter-tâche (i,j)) sont toujours positives. Ceci est suffisant pour satisfaire la condition de qualification sur les arcs sans poser de restrictions sur les coûts c_{ij}. Nous pouvons énoncer ainsi l'algorithme résultant :

Étape 1 : Initialisation

$$Q_i = \begin{cases} \{(0,0)\} & i=p \\ \emptyset & \text{pour tout } i \in N,\ i \neq p \end{cases}$$

Q_i est l'ensemble des étiquettes permanentes au noeud i;

$P_i = \emptyset$ pour tout $i \in N$

P_i est l'ensemble des étiquettes temporaires au noeud i;

Étape 2 : Choix de l'étiquette à traiter

Trouver l'étiquette (T_i^k, C_i^k) de coût lexicographique minimum parmi l'ensemble $T = \bigcup_{i \in N} (Q_i - P_i)$. Si $T \neq \emptyset$, terminer.

Étape 3 : Traitement de l'étiquette (T_i^k, C_i^k)

Pour tous les successeurs j du noeud i faire

début

si $T_i^k + d_{ij} \leq b_j$ (respect des contraintes d'horaires)

alors $(T_j, C_j) = (\max(a_j, T_i^k + d_{ij}),\ C_i^k + c_{ij})$

$Q_j = \overline{Q_j \cup \{(T_j, C_j)\}}$, l'ensemble des étiquette non dominées de $Q_j \cup \{(T, C_j)\}$

fin.

$$P_i = P_i \cup \{(T_i^k, C_i^k)\}$$

aller à l'étape 2.

Le concept de seau généralisé

Le critère de choix de l'étiquette à traiter à l'étape 2 peut être amélioré grâce à l'utisilation d'une généralisation du concept de seau de Denardo et Fox [DENA 79]. Dans l'implantation de l'algorithme de Dijkstra, un seau est une liste de noeuds dont la valeur de l'étiquette est comprise dans un intervalle. Dans le cas simple, le $p^{ième}$ seau est constitué de tous les noeuds dont la valeur de l'étiquette est comprise dans l'intervalle semi-ouvert $[pm,(p+1)m)$ où m est la largeur du seau et est fixé à

$$m = \min_{(i,j)\in A} \{c_{ij}\} .$$

Cette implantation permet de remplacer la recherche de la plus petite étiquette temporaire par la recherche d'un élément quelconque du premier seau à contenir les étiquettes temporaires.

Nous voulons utiliser le même concept de seau en définissant la largeur du seau dans $\mathbb{R}^2$ par

$$(m_d, m_c) = \operatorname*{minlex}_{(i,j)\in A} \{(d_{ij}, c_{ij})\}$$

Soit F(T) la prochaine étiquette à traiter selon l'étape 2 et B(T) le seau généralisé définis par :

$$F(T) = \operatorname*{minlex}_{(T_i^k, C_i^k)\in T} \{(T_i^k, C_i^k)\}$$

$$B(T) = \{(T_i^k, C_i^k) \in T \mid F(T) \overset{L}{<} (T_i^k, C_i^k) \overset{L}{<} F(T) + (m_d, m_c)\}.$$

Nous devons démontrer que de remplacer le traitement de F(T) par le traitement d'un élément quelconque de B(T) ne modifie pas l'optimalité de l'algorithme.

Théorème 4.3. Si $(m_d, m_c) \overset{L}{>} (0,0)$, alors tous les éléments du seau généralisé B(T) ne peuvent être améliorés lors du traitement d'un élément de B(T).

Preuve. Supposons que nous puissions améliorer un des éléments de B(T), (T_j^k, C_j^k) en traitant un autre élément de B(T), (T_i^ℓ, C_i^ℓ)

$$(T_i^\ell, C_i^\ell) + (d_{ij}, c_{ij}) < (T_j^k, C_j^k)$$

$$\rightarrow (T_i^\ell, C_i^\ell) + (d_{ij}, c_{ij}) \overset{L}{<} (T_j^k, C_j^k).$$

Par ailleurs, $(T_j^k, C_j^k) \overset{L}{<} F(T) + (m_d, m_c)$ car $(T_j^k, C_j^k) \in B(T)$

$$\rightarrow (T_i^\ell, C_i^\ell) + (d_{ij}, c_{ij}) \overset{L}{<} F(T) + (m_d, m_c)$$

$$\rightarrow (T_i^\ell, C_i^\ell) \overset{L}{<} F(T) + (m_d, m_c) - (d_{ij}, c_{ij})$$

$$\rightarrow (T_i^\ell, C_i^\ell) \overset{L}{<} F(T) \text{ car } (m_d, m_c) \overset{L}{<} (d_{ij}, c_{ij})$$

ce qui contredit la définition de B(T). Il est donc impossible d'améliorer les éléments de B(T). □

Nous pouvons, lors du choix de l'étiquette à traiter, rechercher un élément quelconque du premier seau B(T) contenant des étiquettes temporaires. Ceci permet de ne calculer ou de n'approximer F(T) que lorsque le seau B(T) est vide. L'étape 2 de l'algorithme devient :

Étape 1'. Trouver le seau actuel

. Trouver $F(T) = \underset{(T_i^k, C_i^k) \in T}{\text{minlex}} \{(T_i^k, C_i^k)\}$

. Calculer la borne supérieure de B(T).

Étape 2. Trouver la prochaine étiquette à traiter

. Trouver un élément de B(T).

Dans ce cas du PPCCCH, les durées d_{ij} étant strictement positives, l'ordre de traitement dépend de la valeur de tmin qui est fixé ainsi:

$$tmin = \min_{(i,j)\in A} d_{ij}.$$

La figure 4.2 permet d'illustrer la fenêtre de temps associée à chaque noeud. Les noeuds sont triés en ordre croissant des débuts de fenêtres de temps; bien que ce ne soit pas nécessaire pour l'implantation de l'algorithme, ceci peut améliorer le comportement de l'algorithme pour les gros problèmes, comme nous le verrons à la section sur les résultats numériques.

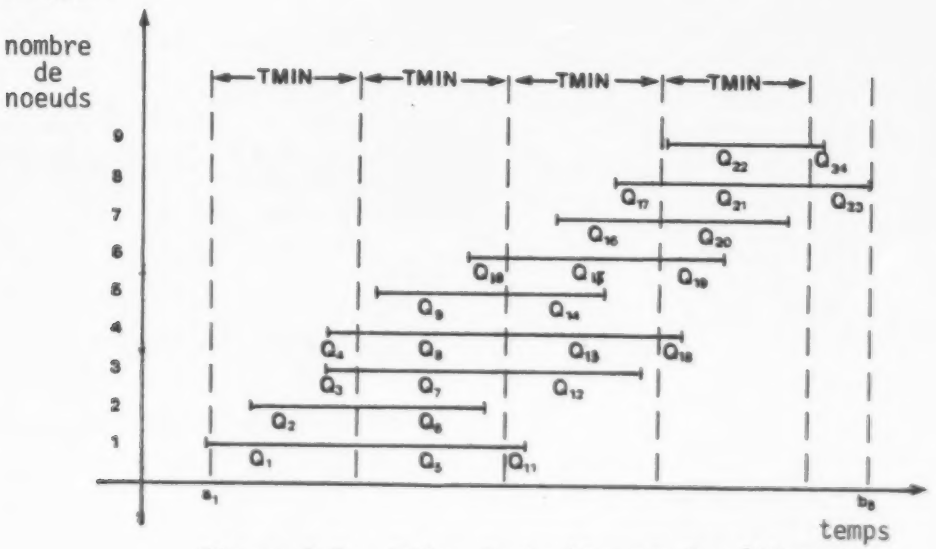


Figure 4.2 : Ordre de traitement des étiquettes selon leurs positions dans les fenêtres de temps.

L'exemple illustré comporte cinq seaux. Nous avons divisé les fenêtres de temps et les étiquettes associées à chaque noeud en sous-ensembles selon les frontières des seaux. Il en résulte 24 sous-ensembles Q_i pouvant chacun contenir des étiquettes. L'ordre de traitement des étiquettes est déterminé par leur appartenance à un sous-ensemble. Le traitement du premier seau débute par le traitement de toutes les étiquettes de Q_1. Il se poursuit par le traitement des étiquettes de Q_2 et ainsi de suite jusqu'à la fin du traitement des étiquettes de Q_4. Nous débutons alors le traitement du second seau par Q_5. L'algorithme se termine par le traitement de Q_{24}.

Réduction et test de dominance

Lors de l'étape 3, nous ajoutons (T_j,C_j) à l'ensemble Q_j. Pour obtenir le nouvel ensemble Q_j^* constitué d'étiquettes non-dominées, nous devons comparer (T_j,C_j) aux étiquettes (T_j^k,C_j^k), k=1,...,K(i) composant Q_j. Les étiquettes sont triées en ordre lexicographique. Puisque les étiquettes permanentes ne peuvent être améliorées, la comparaison est effectuée en débutant par la première étiquette temporaire.

Lors de la comparaison de (T_j,C_j) avec $(T_j^k,C_j^k)\varepsilon Q_j$, quatre cas sont possibles :

1) $(T_j,C_j) < (T_j^k,C_j^k)$: (T_j^k,C_j^k) est éliminée. (T_j,C_j) peut aussi dominer les étiquettes (T_j^ℓ,C_j^ℓ) telles que $\ell > k$.

2) $(T_j,C_j) > (T_j^k,C_j^k)$: (T_j,C_j) est éliminée.

3) $(T_j,C_j) \overset{L}{<} (T_j^k,C_j^k)$: (T_j,C_j) n'est pas dominée, mais elle peut être dominée par les étiquettes permanentes (T_j^ℓ,C_j^ℓ) telles que $\ell < k$.
et $(T_j,C_j) < (T_j^k,C_j^k)$

4) $(T_j,C_j) \overset{L}{>} (T_j^k,C_j^k)$: (T_j,C_j) n'est pas dominée, mais elle peut l'être par les étiquettes (T_j^ℓ,C_j^ℓ) telles que $\ell > k$.
et $(T_j,C_j) > (T_j^k,C_j^k)$

Diverses implantations sont possibles pour les listes d'étiquettes, ce qui influence la façon de réaliser le test de dominance. Nous examinerons quelques unes de ces implantations lors de la prochaine section.

4.6 LA COMPLEXITÉ DE L'ALGORITHME

La complexité de l'algorithme présenté est dépendante du nombre d'étiquettes possibles puisque nous devrons, potentiellement, tenter de classer chacune de ces étiquettes en une seule fois. La complexité dépend aussi de la structure de données utilisée pour implanter les ensembles Q_i. Nous examinerons trois implantations différentes présentant chacune

des caractéristiques la rendant intéressante dans des circonstances particulières.

Soient $D = \sum_{i \in N} (b_i - a_i + 1)$: le nombre d'étiquettes possibles,

$d = \max_{i \in N} (b_i - a_i + 1)$: la plus grande fenêtre,

$n = |N|$: le nombre de noeuds,

$m = |A|$: le nombre d'arcs.

L'implantation réalisée utilise une liste pointée triée en ordre lexicographique pour représenter Q_i. L'espace mémoire utilisé est en pire cas O(m+D). Toutefois, les listes ne sont efficaces que si une faible proportion des étiquettes existe, permettant de réduire la mémoire nécessaire et le travail à effectuer.

L'étape 1 est exécutée une fois et est d'ordre n. L'étape 1' est exécutée au maximum D fois dans le cas où chaque seau ne contient qu'une étiquette et chaque exécution demande au maximum O(n) opérations. L'étape 2 est, dans le même pire cas, exécutée au maximum D fois et trouver un élément de B(T) demande de vérifier les débuts des listes d'étiquettes (O(n) opérations). Le traitement de l'étiquette choisie (étape 3) est effectuée au maximum D fois et le nombre de successeurs pour l'étiquette traitée est au maximum de n. Pour chacun de ces successeurs, la fusion et la réduction à un ensemble d'étiquettes non-dominées associées au noeud i demande la comparaison de la nouvelle étiquette avec au maximum les $b_i - a_i + 1$ étiquettes présentes. Globalement, pour tous les successeurs, cette opération demande O(D) opérations. La complexité de cette implantation est d'ordre $O(D^2)$. Toutefois, le travail à effectuer est proportionnel au carré du nombre d'étiquettes réellement présentes dans la solution. Cette implantation est intéressante si ce nombre est faible par rapport à D.

Les deux autres implantations utilisent une table pour représenter les ensembles Q_i. La table comprend le coût minimum des chemins et ce pour chaque noeud et chaque heure possible d'arrivée à ce noeud. Cette table comprend tous les chemins efficaces ainsi que certains chemins dominés. Pour effectuer la réduction, il suffit d'effectuer à la fin de l'algorithme un test de dominance à chaque noeud. Ce test est de complexité d'ordre O(d). Ceci nous permet, lors de l'étape 3, de vérifier seulement si le coût du nouveau chemin est inférieur à celui de l'ancien chemin. Ces deux implantations utilisent $\Omega(m+D)$ mots de mémoire. Ce sont des implantations efficaces lorsque le nombre d'étiquettes est grand par rapport à D.

Pour les graphes denses, l'utilisation d'une table permet de réaliser une implantation semblable à la première décrite. L'utilisation d'une table permet d'obtenir, grâce à la modification de l'étape 3, une complexité d'ordre O(nD) pour l'algorithme.

La troisième implantation, efficace pour les graphes peu denses, utilise, en plus de la table, un tas (heap) pour y accéder. Le tas permet d'accéder, en temps constant, à l'étiquette lexicographiquement minimum. L'utilisation du tas demande des modifications à l'algorithme. À l'étape 2, lors de la recherche de l'étiquette à traiter, nous obtenons, en temps constant, l'étiquette minimale lexicographiquement. À l'étape 3, le traitement d'une étiquette est modifié puisque nous devons, en plus de vérifier la dominance dans la table, remettre en ordre le tas, ce qui demande O(log nd) opérations. Puisque l'étape 3 est exécutée au maximum nd fois et qu'elle demande l'examen d'au plus md successeurs, la complexité globale de cette étape, et de l'algorithme, est d'ordre O(md log nd) opérations.

Nous pouvons comparer ces trois résultats pour le problème avec arcs lexicographiquement positifs avec celui de Jaffe [JAFF 84] pour le problème avec coûts et durées positifs. Son algorithme, utilisant des tables d'ordre O(nD) mots de mémoire, a une complexité de $O(n^3d)$.

Tableau 4.1

Sommaire de la complexité des algorithmes

implantation	espace-mémoire	temps
avec liste pointée	$O(m+D) \leq O(m+nd)$	$O(D^2) \leq O(n^2d^2)$
avec table	$\Omega(m+D) \leq \Omega(m+nd)$	$O(nD) \leq O(n^2d)$
avec table et tas	$\Omega(m+D) \leq \Omega(m+nd)$	$O(md \log(nd))$
JAFFE	$\Omega(nD) \leq \Omega(n^2d)$	$O(n^3d)$

4.7 RÉSULTATS NUMÉRIQUES

L'algorithme de marquage permanent généralisé (AMPG) présenté à la section 5 a été implanté en FORTRAN 77 en utilisant la première implantation et testé sur un ordinateur CDC CYBER 173 pour comparer ses performances à celles de l'algorithme PCCCH. La comparaison entre les deux algorithmes s'effectuera en variant quatre paramètres :

1. Le nombre de noeuds n composant le réseau. Nous résoudrons des problèmes de 100, 250, 500, 1000 et 2500 noeuds.
2. Le nombre moyen d'arcs par noeud. Nous choisirons aléatoirement parmi les arcs réalisables selon (4.1) de façon à obtenir au total environ 10*n, 25*n, 50*n or 100*n arcs dans le réseau.
3. La largeur des fenêtres de temps. La fenêtre de chaque noeud prendra une largeur de 10, 25, 50 ou 100 unités de temps avec des débuts de tâches a_i distribués entre 0 et 100 selon une loi uniforme.
4. La fraction des noeuds accessible de la source fictive. Les 0.1*n, 0.25*n, 0.5*n ou n noeuds pouvant débuter un chemin sont ceux sans prédécesseur ou dont les a_i sont minimaux.

Les trois premiers paramètres déterminent surtout les caractéristiques des problèmes à résoudre (nombre de noeuds $|N|$, d'arcs $|A|$ et de solutions efficaces $|E_s|$) et le quatrième paramètre influence surtout l'initialisation de l'algorithme.

Trois problèmes tests différents ont été générés pour chaque combinaison réalisable des trois premiers paramètres. Ces problèmes ont été produits en utilisant la même méthode que Desrosiers, Pelletier et Soumis [DESR 83].

Les problèmes ont été produits en utilisant les paramètres suivants:

1. les tâches sont dispersées dans un carré de [0,70] x [0,70] selon une loi uniforme;
2. les durées des tâches d_i sont dispersées dans l'intervalle [5,15] selon une loi uniforme;
3. le temps inter-tâches t_{ij} est égal à la distance euclidienne entre les tâches i et j augmentée de la durée de la tâche i;
4. le coût inter-tâches c_{ij} est égal à t_{ij} moins une grande constante (33 333). Cette constante représente la valeur marginale d'un noeud lors de la génération de colonnes. La constante choisie rend le coût de presque toutes les routes de trois noeuds ou plus négatif.

L'ordinateur utilisé ne permettait pas de conserver en mémoire centrale la totalité des noeuds, des arcs et des étiquettes. Les noeuds et les étiquettes sont conservés en mémoire centrale, alors que les arcs sont divisés en pages et que l'on ne conserve qu'une seule page en mémoire centrale. Lors d'un défaut de page, i.e. lorsque l'exécution de l'algorithme nécessite des arcs d'une autre page, cette nouvelle page est amenée en mémoire centrale. Ce processus peut se comparer à celui de la mémoire virtuelle sur d'autres ordinateurs. Le temps de gestion de la mémoire est comptabilisé séparément du temps d'exécution de l'algorithme. Le tableau 4.2 illustre la comparaison du temps de gestion de la mémoire entre les deux algorithmes pour des problèmes occupant deux, trois ou cinq pages (selon le nombre d'arcs) et ayant 500 noeuds. On peut constater que le temps de gestion est généralement beaucoup plus grand pour PCCCH que pour AMPG, certains problèmes ne pouvant être résolus par PCCCH à cause de trop nombreux défauts de page.

TABLEAU 4.2 - COMPARAISON DU TEMPS DE GESTION DE LA MEMOIRE

			PCCCH				AMPG		
D	|A|	$\|N_d\|$	# DE PROBLEMES RESOLUS	# DE REIN-SERTIONS	TEMPS MEMOIRE (SEC)	TEMPS EXECUTION (SEC)	# DE PROBLEMES RESOLUS	TEMPS MEMOIRE (SEC)	TEMPS EXECUTION (SEC)
5500	12487	50	3	1525	34.30	1.15	3	.43	.99
5500	12487	125	2	838	16.78	1.28	3	.31	1.26
5500	12487	250	3	200	.24	1.37	3	.29	1.31
5500	12487	500	3	3	.32	1.34	3	.29	1.34
5500	24982	50	3	2591	46.41	2.86	3	.99	1.83
5500	24982	125	3	1046	16.88	2.65	3	.57	2.60
5500	24982	250	3	261	.49	2.76	3	.57	2.62
5500	24982	500	3	6	.65	2.49	3	.60	2.59
5500	49984	50	3	3947	43.67	8.99	3	1.89	3.86
5500	49984	125	3	1701	15.75	6.08	3	1.29	5.15
5500	49984	250	3	240	.59	5.58	3	1.22	5.30
5500	49984	500	3	10	.99	5.39	3	1.23	5.27
13000	12527	50	3	1989	45.78	2.21	3	.83	1.53
13000	12527	125	3	1051	29.51	1.77	3	.47	2.21
13000	12527	250	3	438	2.49	2.38	3	.46	2.32
13000	12527	500	3	151	2.14	2.18	3	.49	2.28
13000	24907	50	3	3476	62.65	6.92	3	1.91	3.50
13000	24907	125	3	2088	41.73	4.85	3	1.28	4.84
13000	24907	250	3	729	5.86	5.52	3	1.23	4.96
13000	24907	500	3	244	6.79	4.71	3	1.13	5.05
13000	49988	50	3	4160	88.29	18.41	3	3.42	8.16
13000	49988	125	3	1905	47.78	11.49	3	2.33	10.49
13000	49988	250	3	707	12.11	12.33	3	2.17	10.90
13000	49988	500	3	355	18.36	10.02	3	2.30	10.73
25500	12501	50	3	3374	125.89	5.19	3	1.52	3.06
25500	12501	125	3	1981	86.73	4.46	3	1.61	4.06
25500	12501	250	3	1096	23.90	5.54	3	1.69	4.18
25500	12501	500	3	741	14.55	5.15	3	1.55	4.29
25500	24975	50	3	4702	240.27	9.75	3	3.26	7.69
25500	24975	125	3	2862	105.05	15.27	3	3.23	9.65
25500	24975	250	3	1388	48.76	12.59	3	3.49	9.47
25500	24975	500	3	1064	43.50	11.87	3	3.40	9.66
25500	49953	50	0	0	.00	.00	3	6.73	17.95
25500	49953	125	3	3916	159.22	40.04	3	6.22	21.35
25500	49953	250	3	1937	86.53	31.00	3	6.35	21.27
25500	49953	500	3	1339	93.35	26.18	3	6.71	21.04
50500	12477	50	1	4511	152.11	18.31	3	2.74	6.99
50500	12477	125	2	3623	106.94	19.27	3	1.93	8.80
50500	12477	250	3	2553	79.60	15.62	3	1.65	9.85
50500	12477	500	3	2477	38.75	17.86	3	1.72	9.91
50500	25074	50	0	0	.00	.00	3	6.12	18.58
50500	25074	125	2	4996	306.25	44.77	3	5.07	21.53
50500	25074	250	3	4483	199.50	50.34	3	4.71	22.67
50500	25074	500	3	3710	159.89	43.97	3	4.53	22.91
50500	49913	50	0	0	.00	.00	3	11.70	49.79
50500	49913	125	0	0	.00	.00	3	9.14	57.62
50500	49913	250	1	4575	218.48	165.76	3	8.29	58.16
50500	49913	500	3	4267	323.86	120.73	3	7.71	58.64

TABLEAU 4.3 - STATISTIQUES D'EXÉCUTION DES DEUX ALGORITHMES (1ère partie)

				PCCCH						AMPG					
|N|	|D|	|A|	$\|E_s\|$	# de problèmes résolus	$\|E_T\|$	# de réinsertions	Temps gestion mémoire MOY (sec)	Temps gestion mémoire MAX (sec)	Temps d'exécution MOY (sec)	Temps d'exécution MAX (sec)	# de problèmes résolus	$\|E_T\|$	Temps gestion mémoire MOY (sec)	Temps gestion mémoire MAX (sec)	Temps d'exécution MOY (sec)	Temps d'exécution MAX (sec)
100	1100	1004	149	12	1282	46	-	-	.11	.13	12	1229	-	-	.06	.11
100	1100	2171	184	12	3197	80	-	-	.24	.26	12	3023	-	-	.16	.21
100	2600	963	271	12	1716	93	-	-	.16	.21	12	1599	-	-	.12	.18
100	2600	2520	345	12	6077	129	-	-	.47	.58	12	5421	-	-	.35	.41
100	2600	3232	397	12	8842	142	-	-	.66	.86	12	7761	-	-	.47	.56
100	5100	973	456	12	2870	180	-	-	.30	.37	12	2433	-	-	.23	.28
100	5100	2482	669	12	11650	287	-	-	1.16	1.33	12	9029	-	-	.70	.83
100	5100	4988	917	12	34750	321	-	-	3.33	4.51	12	24813	-	-	1.86	2.17
100	5100	5285	896	12	36034	264	-	-	3.30	3.73	12	26003	-	-	1.94	2.33
100	10100	991	1001	12	7348	412	-	-	1.00	1.13	12	5118	-	-	.55	.66
100	10100	2524	1300	12	33307	737	-	-	4.39	6.61	12	18248	-	-	1.65	1.98
100	10100	5035	1713	12	117341	1118	-	-	16.23	24.07	12	48752	-	-	4.33	5.71
100	10100	8725	2032	12	226901	775	-	-	26.29	32.63	12	101954	-	-	8.94	11.25
250	2750	2521	399	12	3356	153	-	-	.30	.34	12	3215	-	-	.27	.36
250	2750	6249	492	12	10156	322	-	-	.74	1.00	12	9090	-	-	.64	.75
250	2750	12478	578	12	24611	532	7.69	24.52	1.31	1.45	12	20642	.35	.50	1.17	1.39
250	2750	13636	590	12	27424	549	5.43	17.38	1.49	1.77	12	22947	.32	.53	1.32	1.47
250	6500	2510	657	12	4403	231	-	-	.42	.50	12	4139	-	-	.39	.49
250	6500	6220	955	12	17000	501	-	-	1.39	1.88	12	14595	-	-	1.08	1.23
250	6500	12460	1153	12	45080	831	17.17	48.75	2.67	3.08	12	36250	.65	1.09	2.25	2.67
250	6500	20312	1322	12	83844	699	5.30	14.48	5.63	8.30	12	68097	.65	1.06	4.11	4.91
250	12750	2537	1170	12	7344	397	-	-	.80	.96	12	6399	-	-	.68	.81
250	12750	6200	1752	12	30115	839	-	-	3.29	4.22	12	23288	-	-	2.00	2.41
250	12750	12561	2178	12	87234	1084	43.39	107.66	6.68	7.85	12	61075	1.04	1.44	4.63	5.30
250	12750	25018	2691	12	245666	1603	62.38	170.67	20.14	40.97	12	151775	2.47	3.66	10.91	12.72
250	25250	2430	2148	12	16220	996	-	-	2.24	2.55	12	11541	-	-	1.31	1.55
250	25250	6241	3535	12	88171	2209	-	-	13.66	18.67	12	48878	-	-	4.69	5.41
250	25250	12585	4225	12	292451	3751	132.23	300.94	39.15	73.66	12	122300	2.36	3.09	10.61	14.52
250	25250	25092	5231	10	627821	3150	152.61	306.18	78.13	139.06	12	287493	5.69	7.21	25.44	34.05

TABLEAU 4.3 - STATISTIQUES D'EXÉCUTION DES DEUX ALGORITHMES (2e partie)

				PCCCH							AMPG					
							Temps gestion mémoire		Temps d'exécution				Temps gestion mémoire		Temps d'exécution	
|N|	|D|	|A|	$\|E_s\|$	# de problèmes résolus	$\|E_T\|$	# de réinsertions	MOY (sec)	MAX (sec)	MOY (sec)	MAX (sec)	# de problèmes résolus	$\|E_T\|$	MOY (sec)	MAX (sec)	MOY (sec)	MAX (sec)
500	5500	5044	793	12	6820	361	–	–	.60	.68	12	6421	–	–	.55	.62
500	5500	12487	959	11	20745	624	12.56	39.79	1.28	1.42	12	18565	.33	.47	1.22	1.41
500	5500	24982	1071	12	49423	976	16.11	54.34	2.69	3.56	12	41237	.68	1.13	2.41	2.76
500	5500	49984	1181	12	117122	1475	15.25	59.39	6.51	9.63	12	91331	1.41	1.97	4.89	5.60
500	5500	4995	1333	12	9371	512	–	–	.90	.99	12	8632	–	–	.80	.93
500	13000	12527	1757	12	32955	907	19.98	47.13	2.14	2.43	12	28464	.56	.93	2.08	2.39
500	13000	24907	2141	12	88074	1634	29.26	66.20	5.50	7.23	12	71149	1.39	2.22	4.59	5.25
500	13000	49988	2488	12	216772	1782	41.64	108.07	13.06	21.71	12	167179	2.56	3.57	10.07	11.21
500	25500	5033	2391	12	15161	804	–	–	1.73	1.95	12	13057	–	–	1.39	1.63
500	25500	12501	3521	12	63737	1798	62.77	142.76	5.08	6.03	12	47805	1.39	1.94	3.90	4.43
500	25500	24975	4292	11	175573	2304	97.50	265.40	12.61	17.48	12	120484	3.34	3.91	9.12	10.72
500	25500	49953	4939	9	433938	2397	113.03	206.95	32.41	48.69	12	284001	6.50	7.45	20.40	22.81
500	25500	4952	4654	9	35828	1974	–	–	5.11	5.93	12	24598	–	–	2.84	3.59
500	50500	12477	6600	8	154936	3037	80.18	152.11	17.71	21.32	12	93566	2.01	3.06	8.89	10.66
500	50500	25066	8207	7	468973	4278	213.02	395.16	46.02	61.45	12	235377	5.11	6.60	21.42	24.57
500	50500	49913	10682	4	1271690	4344	297.52	357.68	131.98	165.76	12	611655	9.21	13.23	56.05	62.81
1000	11000	9993	1580	12	13282	659	18.91	135.30	1.16	1.32	12	12618	.14	.44	1.09	1.21
1000	11000	25015	1868	12	41005	1378	23.46	77.33	2.61	2.91	12	36063	.77	1.01	2.49	2.79
1000	11000	50129	2106	7	93420	1303	59.92	242.48	5.59	7.10	12	81026	1.90	2.59	4.98	5.67
1000	11000	99739	2438	8	220494	1885	39.00	196.74	11.91	14.06	12	187361	3.74	5.17	10.71	12.23
1000	11000	10018	2637	7	18431	760	8.42	36.97	1.68	1.92	12	16774	.42	.77	1.62	1.88
1000	26000	25058	3745	12	69955	2014	45.17	146.62	4.43	5.30	12	60312	1.17	1.78	4.73	5.41
1000	26000	50147	4417	9	173245	2562	76.92	190.97	10.04	13.81	12	142898	2.61	3.93	10.23	11.13
1000	26000	100103	5225	9	446126	3453	115.12	300.26	26.60	43.25	12	347727	6.19	9.20	23.03	24.89
1000	51000	10017	4723	8	30815	1406	8.62	42.06	3.24	3.67	12	25856	.58	1.16	2.93	3.34
1000	51000	25080	6895	12	127529	3607	171.00	377.51	8.46	14.39	12	95857	2.47	3.58	8.82	9.80
1000	51000	50052	8607	7	331961	3033	166.82	191.25	19.73	30.76	12	240228	6.95	9.86	19.94	22.53
1000	51000	99959	10720	4	856061	3237	289.56	386.90	49.97	63.86	12	612340	16.66	17.32	47.62	53.62
1000	101000	10012	9224	7	71637	3879	.85	4.09	10.25	11.75	12	48816	1.54	2.87	5.81	6.97
1000	101000	25065	12557	0	–	–	–	–	–	–	5	174509	5.67	7.49	17.33	19.72
1000	101000	50051	13699	0	–	–	–	–	–	–	1	407611	11.98	11.98	37.79	37.79
2500	27500	24957	3907	0	–	–	–	–	–	–	12	31363	2.20	3.00	2.76	3.11
2500	27500	62446	4705	0	–	–	–	–	–	–	12	90001	7.42	10.42	6.25	6.95
2500	27500	124902	5291	0	–	–	–	–	–	–	12	204891	14.35	19.12	13.05	14.93
2500	27500	249558	6033	0	–	–	–	–	–	–	12	462105	33.36	40.82	27.24	30.60
2500	65000	25060	6977	0	–	–	–	–	–	–	12	42551	3.30	4.30	4.06	4.70
2500	65000	62503	8944	0	–	–	–	–	–	–	12	143481	12.50	14.12	11.03	12.10
2500	65000	125196	10709	0	–	–	–	–	–	–	10	356271	25.85	31.79	24.98	27.86

Il est à noter que le temps de gestion de la mémoire de PCCCH dépend du nombre de noeuds pouvant débuter une route $|N_d|$. Lors de l'initialisation de PCCCH, la liste des noeuds à traiter est la liste des noeuds de N_d en ordre croissant de début des fenêtres de temps. Puisque c'est aussi l'ordre de classement des noeuds dans les pages, ceci permet d'expliquer que plus la liste initiale des noeuds à traiter est grande, plus l'ordre des noeuds dans les pages produisant ainsi moins de réinsertions et moins de défaut de page. L'ordre de traitement défini à la section 5 permet à AMPG de définir un ordre de traitement entraînant peu de défauts de page.

Le tableau 4.3 contient diverses statistiques sur l'exécution des deux algorithmes. L'algorithme AMPG a résolu tous les problèmes sauf ceux dont la solution comportait trop d'étiquettes efficaces excédant ainsi la quantité de mémoire disponible. L'algorithme PCCCH n'a pu résoudre, à cause du trop grand nombre de défauts de page, certains problèmes que AMPG avait résolus; ces problèmes représentent 1% des problèmes de 250 noeuds, 13% des problèmes de 500 noeuds et 30% des problèmes résolus de 1000 noeuds. Prévoyant que PCCCH ne pourrait résoudre que moins de 25% des problèmes résolus de 2500 noeuds, ces problèmes n'ont pas été exécutés avec PCCCH. La présence de problèmes non-résolus complique la comparaison des algorithmes, les problèmes non-résolus étant les plus difficiles.

Les trois variables exposées dans le tableau 4.3 sont le nombre d'étiquettes temporaires produites $|E_T|$, le temps de gestion de la mémoire et le temps d'exécution de l'algorithme. L'algorithme AMPG garantit, en ne traitant que les successeurs de solutions efficaces, que le nombre d'étiquettes temporaires est minimal. À cause des réinsertions possibles, l'algorithme PCCCH peut être amené à produire plus d'étiquettes temporaires, dans plusieurs cas, on note une augmentation de 50% par rapport au nombre d'étiquettes temporaires par AMPG avec certains cas où l'augmentation est de plus de 100%.

Le temps d'exécution global des algorithmes comprend deux parties : le temps d'exécution de l'algorithme et le temps de gestion de la mémoire. L'algorithme AMPG domine clairement l'algorithme PCCCH pour le temps global, cette domination est due surtout à une meilleure gestion de la mémoire et cela demeure vrai par tous les ordinateurs fonctionnant avec une mémoire limitée ou une mémoire virtuelle. Cet aspect devient particulièrement important pour de grands problèmes.

Les résultats plus détaillés du tableau 4.2 permettent de constater que dans certains cas précis, l'algorithme PCCCH est aussi, et parfois plus, efficace que AMPG. Les facteurs les plus importants pour une bonne performance de PCCCH sont :

1. une initialisation complète de la liste des noeuds à traiter;
2. peu de solutions efficaces par noeud (moins de 5).

Ces deux conditions permettent d'avoir peu de réinsertions et rendent les temps d'exécution de PCCCH comparables à ceux de AMPG. Si les deux conditions ne sont pas remplies, PCCCH demande un temps d'exécution qui, dans plusieurs cas, est deux fois celui de AMPG. Dans un contexte de génération de colonnes, l'algorithme choisi pourra être exécuté plusieurs centaines de fois, rendant toute différence de performance très significative.

4.8 CONCLUSION

L'utilisation d'un théorème sur l'ordre de traitement des étiquettes et du concept de seau généralisé permet la mise au point d'un algorithme (AMPG) pour résoudre en temps pseudo-polynomial le problème de chemin le plus court avec contraintes d'horaires. Ce nouvel algorithme peut aussi être utilisé pour résoudre des exemplaires de chemins les plus courts avec une contrainte supplémentaire ou avec double inégalité. Au prix d'une hypothèse supplémentaire sur les coûts, il serait aussi

possible d'utiliser cet algorithme pour trouver les solutions efficaces d'un problème bicritère.

L'algorithme AMPG permet de résoudre efficacement tous les exemplaires du PPCCCH, au contraire de PCCCH qui possède d'assez nombreux mauvais problèmes, tel que vu lors de l'analyse des résultats numériques.

CHAPITRE V

UN ALGORITHME DE RÉOPTIMISATION POUR LE PROBLÈME DE PLUS COURT CHEMIN AVEC CONTRAINTES D'HORAIRES

5.1 LE PROBLÈME

Le problème du plus court chemin avec contraintes d'horaires consiste à trouver le chemin élémentaire de coût minimum entre deux noeuds en débutant la visite de chaque noeud i choisi à l'intérieur de sa fenêtre de temps $[a_i,b_i]$. Soit le réseau G=(N,A) où l'ensemble des noeuds N est constitué des tâches, d'une source p et d'une destination q. À chaque arc (i,j) est associé un coût c_{ij} négatif ou positif comprenant le coût inter-tâches (i,j) ainsi que le coût de la tâche i et une durée d_{ij} positive comprenant la durée inter-tâches (i,j) et la durée de la tâche i. Étant donné les fenêtres de temps $[a_i,b_i]$ pour le début de chaque tâche, tous les arcs $(i,j)\in A$ permettent, en débutant la tâche i au début de sa fenêtre de temps, de débuter l'exécution de la tâche j avant la fin de sa fenêtre de temps, i.e. tous les arcs respectent la condition

$$a_i + d_{ij} \leq b_j \tag{5.1}$$

Nous pouvons formuler le problème de plus court chemin élémentaire avec contraintes d'horaires (PPCCCH) de la façon suivante :

$$\text{Minimiser} \quad c.x = \sum_{(i,j)\in A} c_{ij}.x_{ij} \tag{5.2}$$

sujet à

$$x\in X(G) = \begin{cases} \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = \begin{cases} +1 & i=p \\ 0 & i\in N, i\neq p, i\neq q \\ -1 & i=q \end{cases} & (5.3) \\ x_j \geq 0 \quad \text{pour tout } (i,j)\in A & (5.4) \end{cases}$$

$$x_{ij} > 0 \Rightarrow t_i + d_{ij} \leq t_j \qquad (i,j)\in A \tag{5.5}$$

$$a_i \leq t_i \leq b_i \qquad i\in N, \tag{5.6}$$

où x_{ij} représente le flot sur l'arc (i,j) et t_i, le temps de début de la tâche i.

Ce problème apparaît comme sous-problème lors de la fabrication de routes de véhicules. Le problème global consiste à déterminer le nombre et la composition des routes permettant de réaliser toutes les tâches de façon à minimiser le coût total. Desrosiers, Soumis et Desrochers [DESR 84a] utilisent la méthode de génération de colonnes pour construire des routes recouvrant de façon optimale l'ensemble des tâches à effectuer. À chaque itération de la méthode de génération de colonnes, la méthode cherche à améliorer la solution actuelle en insérant dans la base des routes de coût marginal négatif comprenant des sous-ensembles disjoints de tâches respectant les contraintes d'horaires.

Le calcul de plusieurs chemins disjoints recouvrant l'ensemble des noeuds du réseau demande la résolution d'une suite de PPCCCH dont chacun calcule un des chemins. Après avoir solutionné le $k^{ième}$ problème pour un ensemble de noeuds N^k, les noeuds S^k appartenant à la solution optimale X^k sont enlevés de N^k pour former un ensemble de noeuds N^{k+1} sur lesquels portera le $k+1^{ième}$ problème. Les algorithmes actuels ne permettent pas de profiter des solutions efficaces déjà calculées pour accélérer le calcul de la nouvelle solution optimale.

Le retrait des noeuds de la solution S^k pour obtenir le $k+1^{ième}$ problème correspond en fait à une grande augmentation du coût des arcs adjacents aux noeuds de S^k. La méthode proposée pour la réoptimisation après retrait de noeuds pourra aussi être utilisée pour réoptimiser après avoir modifié les coûts sur les arcs.

5.2 REVUE DE LA LITTÉRATURE

La réoptimisation du problème de plus court chemin a déjà été étudiée. Le premier résultat théorique a été obtenu par Spira et Pan [SPIR75]

qui ont démontré que la complexité de la réoptimisation dans un graphe où les coûts sont non-négatifs ne peut être meilleure que celle de l'optimisation.

Les cas de réoptimisation du problème de plus court chemin les plus étudiés sont : 1) une diminution du coût d'un ou plusieurs arcs. Radionov [RADI 68] a décrit les équations à la base de cette réoptimisation et a présenté un algorithme. Goto et Sangiovanni-Vincentelle [GOTO 78] ont présenté un nouvel algorithme utilisant moins d'espace. Fujishige [FUJI 81] et HAJI [HAJI 81] ont présenté des améliorations à l'algorithme présenté dans [GOTO 78]; 2) une modification quelconque d'un ou plusieurs arcs. Murchland [MURC 67] a présenté le premier algorithme dans le cas de la perturbation d'un seul arc dans un réseau orienté et Dionne [DION 78] l'a amélioré et étendu au cas non orienté. PAPE [PAPE 74] a analysé le cas de perturbation des coûts de plusieurs arcs. Gallo [GALL 80] a présenté un algorithme pour le cas d'une perturbation de coût de plusieurs arcs et l'a appliqué au changement du noeud-source lors du calcul du plus court chemin entre toutes les paires de noeuds.

Plusieurs méthodes ont été mises au point pour la résolution du PPCCCH ou de problèmes similaires. Ces méthodes se divisent en deux familles : les méthodes lagrangiennes qui associent un multiplicateur à chaque contrainte et résolvent des problèmes de plus court chemin, et la programmation dynamique dont un exemple sera détaillé dans cette section. Une revue de la littérature plus complète sur ces deux familles de méthodes a été effectuée au chapitre précédent.

À chaque chemin réalisable X_{pj} de l'origine p à un noeud j est associée une étiquette (temps, coût) correspondant respectivement à l'heure d'arrivée au noeud j et au coût du chemin X_{pj}. Ces étiquettes seront représentées par (T_j^k, C_j^k) dénotant les caractéristiques du $k^{ième}$ chemin X_{pj}^k, les indices k ou j pourront être absents si le contexte n'est pas ambigu.

Le calcul de cette étiquette est effectué itérativement le long du chemin $X_{pj} = (i(0),i(1),\ldots,i(\ell))$ où $i(0) = p$ et $i(\ell) = j$ de la façon suivante :

$$T_{i(1)} = 0$$

$$C_{i(1)} = 0$$

$$T_{i(j)} = \max\{a_{i(j)}, T_{i(j-1)} + d_{i(j-1)i(j)}\} \qquad j=1,\ldots,\ell$$

$$C_{i(j)} = C_{i(j-1)} + c_{i(j-1)i(j)} \qquad j=1,\ldots,\ell$$

Définition 5.1.

Soit X^1 et X^2 deux chemins distincts de p à j et leurs étiquettes associées (T^1,C^1) et (T^2,C^2), X^1 domine X^2 ou $(T^1,C^1) < (T^2,C^2)$ si et seulement si $(T^2,C^2)-(T^1,C^1) > (0,0)$ et $(T^1,C^1)\neq(T^2,C^2)$.

La relation de dominance n'étant pas une relation d'ordre total, elle ne permet pas d'ordonner tous les chemins. La relation de dominance nous permet toutefois de conclure qu'un chemin X_{pj} non-dominé (efficace) est le plus court chemin arrivant au noeud j à l'heure T_j ou plus tôt. Ceci implique la possibilité de plusieurs chemins efficaces pour chaque noeud. Les étiquettes associées à ces chemins forment un ensemble Q_j à chaque noeud. La figure 5.1 illustre la relation entre différentes étiquettes (temps, coût) associées à ces chemins efficaces (X^1 à X^4) et à d'autres dominés (X^5 à X^8).

Cette relation permet de définir le coût du chemin réalisable de coût minimal comme une fonction du temps d'arrivée. La figure 5.1 permet d'illustrer cette fonction pour un noeud.

Les algorithmes de programmation dynamique mis au point pour la résolution du PPCCCH calculent tous les chemins efficaces de la source à tous les noeuds en solutionnant les équations de récurrence suivantes :

$$C(j,t) = \min_{\substack{i\in N \\ a_i<d<b_i}} \{C(i,d)+c_{ij} \mid d+d_{ij}<t,(i,j)\in A\} \text{ pour tout } \underset{a_j<t<b_j}{j\in N}, \qquad (5.7)$$

$$C(p,a_p) = 0 \qquad (5.8)$$

C(i,t) représentant le coût minimum parmi les chemins arrivant au noeud i au temps t ou auparavant. Le chemin efficace de coût minimum au noeud destination est la solution du problème (5.2)-(5.6). Pour calculer la solution, Desrosiers, Pelletier et Soumis [DESR83] proposent une généralisation de l'algorithme de Ford-Bellmam et au chapitre précédent nous avons proposé une généralisation de l'algorithme de Dijkstra.

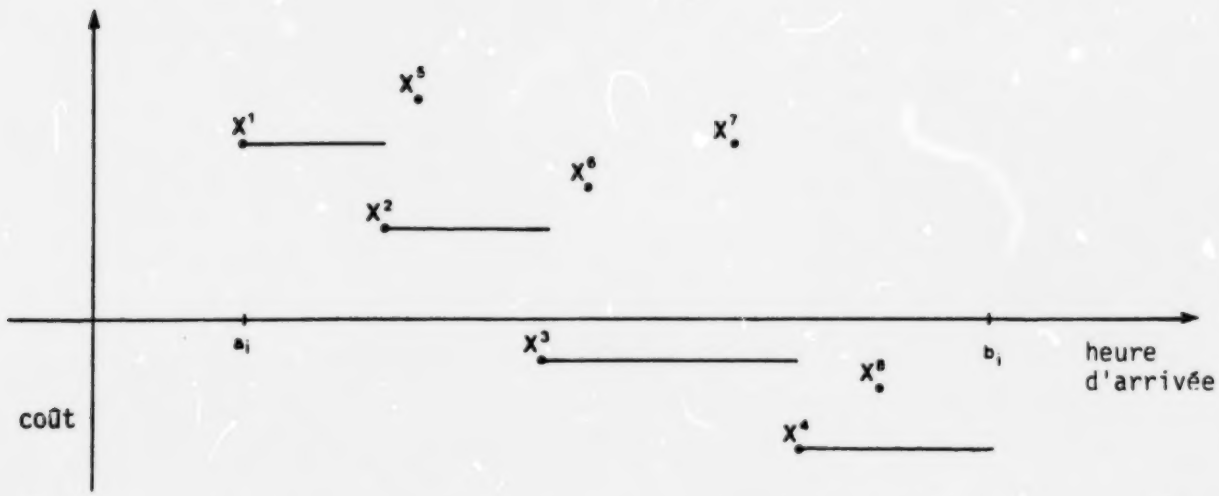


Figure 5.1 : Relation de dominance entre étiquettes associées à différents chemins

5.3 UTILISATION DE L'ANCIENNE SOLUTION

Soit Z^0 le problème pour lequel nous disposons de tous les chemins efficaces et Z^1 le nouveau problème portant sur un ensemble réduit de noeuds $N^1=N^0-S^0$.

Les chemins efficaces X_{pj} de l'ancienne solution pour un noeud $j \in N$ donné se divisent en deux catégories :

1. les chemins réalisables, i.e. dont aucun noeud ne fait partie de S^0. Ces chemins sont des chemins efficaces du problème Z^1.

2. les chemins éliminés, i.e. ayant au moins un noeud faisant partie de S^0. Ces chemins, n'étant plus réalisables, devront être remplacés par l'algorithme de réoptimisation.

L'ensemble des chemins efficaces de Z^0 formant un arbre, il suffit d'éliminer tous les successeurs d'un noeud appartenant à la solution optimale de Z^0, pour déterminer le statut de chaque chemin.

À chaque noeud de N, l'ensemble Q_j des étiquettes associées aux chemins efficaces de Z^0 peut être partitionné en

$$P_j = \{(T_j^k, C_j^k) \mid \text{le chemin associé } X_{pj}^k \text{ est réalisable}\},$$

$$D_j = \{(T_j^k, C_j^k) \mid \text{le chemin associé } X_{pj}^k \text{ est éliminé}\}.$$

Les éléments des ensembles P_j appartenant aux chemins efficaces du problème Z^1, l'obtention d'une solution complète pour Z^1 demande le calcul de remplaçants pour les étiquettes des ensembles D_j. Les étiquettes des D_j sont des bornes inférieures sur les valeurs de leurs remplaçants car dans Z^0, les éléments des D_j dominaient leurs éventuels remplaçants.

La figure 1 définissait, avec les étiquettes d'un ensemble Q_j, le coût d'un chemin efficace X_{pj} comme une fonction du temps d'arrivée. À l'aide des étiquettes de l'ensemble $Q_j = P_j \cup D_j$, associées aux solutions efficaces de Z^0, il est possible de définir une fonction duale représentant une borne inférieure sur le coût des solutions efficaces X_{pj} du problème Z^1 en fonction de l'heure d'arrivée au noeud j. De même, les étiquettes de l'ensemble P_j, associées à des chemins réalisables de Z^0, permettent de définir une fonction primale représentant une borne supérieure du coût des solutions efficaces X_{pj} du problème Z^1 en fonction de l'heure d'arrivée au noeud j. La figure 5.2 illustre ces deux fonctions pour un noeud.

Les zones hachurées dans la figure 5.2 représentent les zones d'incertitude. L'algorithme de réoptimisation devra modifier les ensembles P_j et D_j de façon à remplacer les étiquettes des ensembles D_j par de nouvelles étiquettes réalisables s'ajoutant aux ensembles P_j de façon à éliminer l'incertitude.

5.4 PRINCIPES DE LA RÉOPTIMISATION

Il y a deux façons de réduire la taille de la zone d'incertitude. La première est d'augmenter la fonction duale en remplaçant une étiquette

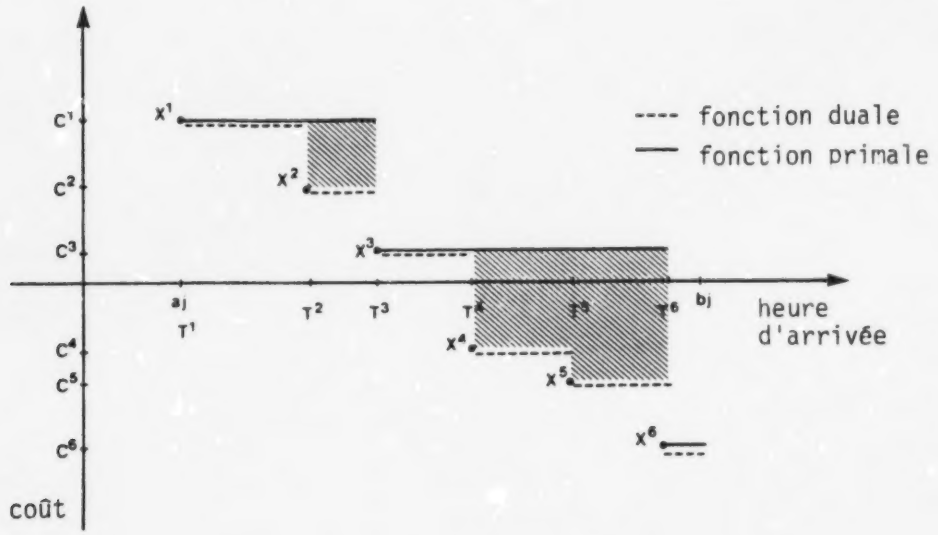


Figure 5.2 : Exemple des fonctions primale et duale à un noeud

de D_j par une nouvelle borne inférieure. La seconde est de diminuer la fonction primale en créant de nouvelles étiquettes réalisables efficaces s'ajoutant à P_j. Ces calculs se limitent à une des zones d'incertitude. Par exemple, si nous voulons réduire la première zone d'incertitude dans la figure 2, située entre T^2 et T^3, les nouvelles étiquettes réalisables ou non devront être comprises dans le rectangle de sommets (T^2,C^1), (T^2,C^2), (T^3,C^1) et (T^3,C^2).

La première façon de réduire l'incertitude est de calculer une nouvelle fonction duale pour la zone délimitée. Cette nouvelle borne inférieure est calculée en trouvant tous les chemins X_{pi} dont le prolongement par un arc $(i,j)\varepsilon A$ produit un chemin X_{pj} ayant son étiquette dans la zone délimitée. Le sous-ensemble EFF des étiquettes efficaces est calculé pour obtenir les étiquettes définissant la borne inférieure

$$R_j = EFF\{(T_j,C_j) = (\max\{a_j, T_i^k+d_{ij}\},\ C_i^k+c_{ij}) \mid$$
$$\text{pour tout } (i,j)\varepsilon A,\ (T_i^k,C_i^k)\varepsilon P_i \cup D_i \text{ et } (T_j,C_j)$$
$$\text{appartient à la zone à explorer}\}.$$

L'autre façon de réduire l'incertitude est d'améliorer la fonction primale par le calcul de solutions réalisables situées dans la zone délimitée. Il n'y a présentement aucune solution réalisable dans la zone d'incertitude. Les solutions réalisables sont calculées en trouvant tous les chemins réalisables X_{pi} dont le prolongement par un arc $(i,j)\varepsilon A$ produit un chemin X_{pj} ayant son étiquette dans la zone délimitée. De façon à calculer

$$RP_j = EFF\{(T_j,C_j) = (\max\{a_j, T_i^k+d_{ij}\}, C_i^k+c_{ij}) \mid$$
$$\text{pour tout } (i,j)\varepsilon A,\ (T_i^k,C_i^k)\varepsilon P_i \text{ et } (T_j,C_j)$$
$$\text{appartient à la zone à explorer}\}$$

l'ensemble des solutions efficaces réalisables formant la nouvelle fonction primale dans la zone délimitée. Certaines des étiquettes de RP_j peuvent être dominées par des étiquettes non-réalisables de R_j.

Le remplacement de l'étiquette éliminée s'effectue

1) en trouvant les nouvelles étiquettes réalisables (εRP_j) pour les ajouter à P_j, Cet ajout d'étiquettes, appartenant à la fois à la borne supérieure et à la borne inférieure, détruit l'incertitude en ce point :

$$P_j = P_j \cup (R_j \cap RP_j)$$

2) en ajoutant les étiquettes non réalisables ($\notin RP_j$) et efficaces ($\in R_j$) à l'ensemble D_j après avoir éliminé l'étiquette remplacée :

$$D_j = (D_j - \{(T_j^k, C_j^k)\}) \cup (R_j - (R_j \cap RP_j)).$$

pour définir la nouvelle borne inférieure.

Lors du remplacement d'une étiquette éliminée, il faut s'assurer que l'étiquette à remplacer ne soit pas produite à nouveau, entraînant une étape sans réduction de la zone d'incertitude. Au chapitre précédent, un ordre de traitement et des conditions de qualification sur les arcs sont définis pour la résolution du PPCCCH par un algorithme primal. Les mêmes notions sont reprises ici pour définir un ordre de remplacement des étiquettes de $D = \bigcup_{j \in N} D_j$ par l'algorithme primal-dual de réoptimisation.

Définition 5.2.

Soit $a,b \in R^2$, a est lexicographiquement plus petit que b ($a \overset{L}{<} b$) si et seulement si $a \neq b$ et que le premier élément non-nul de b-a est positif.

Définition 5.3.

Un arc (i,j) est lexicographiquement posisif si et seulement si $(0,0) \overset{L}{<} (d_{ij}, c_{ij})$.

L'ordre lexicographique est un ordre total dans R^2 compatible avec l'ordre partiel de dominance.

L'utilisation du seau généralisé pour la réoptimisation

Le critère de choix de l'étiquette duale à remplacer doit idéalement permettre de déterminer

1) si une étiquette est prête à être remplacée, i.e. si son remplacement diminue la zone d'incertitude et elle fait partie du seau des étiquettes à remplacer;

2) si une étiquette est prête à être remplacée, est-ce qu'il est possible de retarder son remplacement sans affecter la validité de l'algorithme, i.e. existe-t-il d'autres étiquettes à remplacer dans le seau actuel?

Cette dernière information permet, dans le cas d'un grand réseau où les données relatives à une partie seulement du réseau sont disponibles, de retarder au maximum le traitement des parties non-disponibles du réseau.

Le concept de seau généralisé exposé au chapitre précédent permet de répondre à ces questions.

Soit $(d_j,c_j) = \underset{(i,j)\in A}{\text{minlex}} \{(d_{ij},c_{ij})\}$ et $(m_d,m_c) = \underset{i\in N}{\text{minlex}} \{d_i,c_i)\}$

Soit $F(D) = \underset{(T_i^k,C_i^k)\in D}{\text{minlex}} \{(T_i^k,C_i^k)\}$,

définissons

$$B(D) = \{(T_i^k,C_i^k)\in D \; F(D) \overset{L}{\leq} (T_i^k,C_i^k) \overset{L}{\leq} F(D)+(d_j,c_j)\}$$

Théorème 5.1.

Si $(m_d,m_c) \overset{L}{>} (0,0)$ lors du remplacement d'une étiquette de B(D)

1) il est impossible de reproduire l'étiquette remplacée;
2) la zone d'incertitude diminue.

Preuve.

1) Supposons qu'il soit possible de reproduire l'étiquette à remplacer $(T_j,C_j)\in B(D)$, il existe donc une étiquette $(T_i,C_i)\in D$ telle que

$$(T_i,C_i) + (d_{ij},c_{ij}) = (T_j,C_j) \overset{L}{\leq} F(D) + (d_j,c_j)$$

$$(T_i,C_i) + (d_{ij},c_{ij}) \overset{L}{\leq} F(D) + (d_j,c_j)$$

$$(T_i,C_i) \overset{L}{\leq} F(D) \quad \text{car} \quad (d_{ij},c_{ij}) \overset{L}{>} (d_j,c_j) \quad \text{par définition.}$$

Ce qui contredit la définition de F(D).

2) L'algorithme calcule une nouvelle fonction duale en calculant les nouveaux éléments réalisables et non-réalisables de la borne inférieure pour la zone délimitée de la façon suivante :

$$R_j = \min\{(T_j^\ell, C_j^\ell) = (\max\{a_j, T_i^k + d_{ij}\}, C_i^k + c_{ij}) \mid (T_i^k, C_i^k) \in P_i \cup D_i,\ (i,j) \in A \text{ et } (T_j^\ell, C_j^\ell) \text{ fait partie de la zone délimitée.}\}$$

Les étiquettes réalisables de R_j éliminant l'incertitude en définissant à la fois la fonction duale et la fonction primale, seules les étiquettes non-réalisables de R_j délimitent l'incertitude due à un écart entre les deux fonctions. Une borne inférieure lexicographique sur la valeur de la plus petite étiquette non-réalisable de R_j est :

$$(\max\{a_j, T_j + d_j\},\ C_j + c_j).$$

En effet, $(\max\{a_j, T_j + d_j\},\ C_j + c_j) \overset{L}{\leq} (\max\{a_i, T_i^k + d_{ij}\},\ C_i^k + c_{ij})$ pour tout $(T_i^k, C_i^k) \in D$ et $(i,j) \in A$.

Cette borne inférieure nous garantit que l'écart entre les fonctions duale et primale sera réduit d'au moins $(d_j, c_j) \overset{L}{>} 0$, la zone est donc réduite sur l'axe du temps si $d_j > 0$ ou sur l'axe du coût si $c_j > 0$.

5.5 L'ALGORITHME DE RÉOPTIMISATION

L'utilisation du théorème 5.1 permet la mise au point de l'algorithme suivant pour la réoptimisation du PPCCCH.

Étape 1. Initialisation

À partir de la solution optimale du problème précédent, déterminer les étiquettes éliminées par le retrait des noeuds de la solution précédente.

- préparation des ensembles P_j et D_j.
- calcul des (dj,cj) pour chaque noeud.

Étape 2. Calcul des limites du seau

Trouver l'étiquette (T_j^k, C_j^k) de coût lexicographique minimum parmi l'ensemble $D = \bigcup_{j \in N} D_j$. Si $D = \emptyset$, terminer

Étape 3. Recherche d'une étiquette (T_j^k, C_j^k) à traiter

- Trouver une étiquette appartenant au premier seau.
- Si le seau est vide, retourner à l'étape 2.

Étape 4. Choix de la zone à explorer parmi la zone d'incertitude

Étape 5. Remplacement de l'étiquette (T_j^k, C_j^k)

A) Calcul de la nouvelle fonction duale dans la zone à explorer

$$R_j = \min\{(T_j, C_j) = (\max\{a_j, T_i^k + d_{ij}\},\ C_i^k + c_{ij}) \mid$$

pour tout $(i,j) \in A$, $(T_i^k, C_i^k) \in P_i \cup D_i$

et (T_j, C_j) est dans la zone à explorer.$\}$

B) Calcul de la fonction primale dans la zone à explorer

$$RP_j = \min\{(T_j, C_j) = (\max\{a_j, T_i^k + d_{ij}\}, C_i^k + c_{ij}) \mid$$

pour tout $(i,j) \in A$, $(T_i^k, C_i^k) \in P_i$

et (T_j, C_j) est dans la zone à explorer.$\}$

C) Création des nouveaux ensembles D_j et P_j

- $P_j = P_j \cup (R_j \cap RP_j)$
- $D_j = (D_j - \{(T_j^k, C_j^k)\}) \cup (R_j - (R_j \cap RP_j))$

retourner à l'étape 3.

Lors de la description des principes de réoptimisation, un exemple de zone à explorer a été donné. Toutefois, aucune définition précise n'a été énoncée. Puisque la zone à explorer débute avec l'apparition d'un

"saut de dualité" entre la fonction duale et la fonction primale et qu'elle est délimitée par ces deux fonctions, il ne reste qu'à définir la fin dans le temps de la zone.

Le choix de $TR_j = \min\{T_j^\ell \mid (T_j^\ell, C_j^\ell) \in P_j \text{ et } T_j^\ell < T_j^k\}$ le temps d'arrivée du premier chemin réalisable postérieur à l'étiquette (T_j^k, c_j^k) remplacée définit la zone à explorer de dimension maximale correspondant à la zone d'incertitude complète. La zone ainsi définie peut être très grande par rapport à la réduction de la zone d'incertitude assurée par le théorème 5.1. La construction de la borne supérieure est surtout efficace au début de la zone d'incertitude. L'utilisation d'une zone d'exploration beaucoup plus grande que la réduction d'incertitude assurée peut aussi entraîner une série de remplacements avant d'éliminer complètement l'incertitude. Par exemple, si la durée de la zone est de n et que la diminution assurée est de 1, dans le pire cas, la dernière partie de l'intervalle peut être traité n fois.

Pour éviter des séries de remplacements, il vaut mieux utiliser une longueur fixe basée sur la réduction assurée de l'incertitude ou une longueur variable basée sur cette valeur. Par exemple :

$$TSUP_1 = \min\{T_j^k + \alpha d_j, TR_j\}$$

$$\alpha > 1 \quad \text{et} \quad (d_j, c_j) = \underset{(i,j)\in A}{\text{minlex}} \{(d_{ij}, c_{ij})\}.$$

Ce critère de définition de la zone à explorer garantit qu'aucune partie de la zone d'incertitude ne sera traitée plus de α fois, permettant la mise au point d'un algorithme plus efficace.

Cet algorithme de réoptimisation diffère des autres algorithmes de programmation dynamique pour la résolution du PPCCCH [DESR83] et d'autres algorithmes de programmation dynamique car il n'utilise pas le marquage de successeurs pour résoudre les équations de récurrence (5.7-5.8).

L'algorithme proposé procède plutôt en marquant par les prédécesseurs des des chemins se terminant dans un intervalle de temps exploré et résolvant complètement (5.7-5.8) pour cet intervalle. Selon cet algorithme, et contrairement aux autres algorithmes existants, seule la solution au noeud où une étiquette est remplacée peut être modifiée. La seule mention d'une méthode semblable dans la littérature est la résolution récursive de Denardo et Fox [DENA79] pour le calcul du plus court chemin dans un graphe acyclique ordonné. La méthode proposée, contrairement à celle de Denardo et Fox, détermine l'ordre de traitement des étiquettes.

5.6 COMPLEXITÉ ET IMPLANTATION

Il serait logique de croire que la complexité d'un algorithme de réoptimisation pour le PPCCCH serait moindre que celle de l'algorithme d'optimisation. Spira et Pan [SPIR75] ont démontré que cette présomption est fausse pour le problème de plus court chemin. Puisque le PPCCCH est en fait un plus court chemin dans un graphe augmenté composé de D noeuds et d'au plus dm arcs,

où $D = \sum_{i=1}^{m} (b_i - a_i + 1)$: le nombre d'étiquettes possibles,

$d = \max_{i \in N} \{b_i - a_i + 1\}$: la plus grande fenêtre

$n = |N|$: le nombre de noeuds dans G

$m = |A|$: le nombre d'arcs dans G,

cette présomption est aussi fausse pour PPCCCH. Les gains possibles de temps d'exécution dépendent donc du problème particulier et non de la complexité de l'algorithme de réoptimisation.

Deux implantations sont possibles pour cet algorithme selon la quantité de mémoire centrale disponible pour représenter les étiquettes (états de la programmation dynamique). La première possibilité est de conserver toutes les étiquettes possibles dans une table de dimension D

permettant un accès direct aux étiquettes. La seconde possibilité est de ne conserver que les étiquettes efficaces dans une liste pointée permettant un accès séquentiel aux étiquettes efficaces. Selon les résultats obtenus précédemment, seulement 5% à 20% des étiquettes sont efficaces, et l'utilisation d'une liste permet une économie appréciable de mémoire.

La complexité des trois premières étapes de l'algorithme est indépendante de la structure de données choisie. Globalement ces trois étapes demandent O(m+D+Dlog n) opérations.

L'initialisation lors de l'étape 1 prépare les ensembles P_i et D_i en vérifiant le statut (réalisable ou éliminé) du précédesseur de chaque étiquette. Le statut de chaque étiquette étant déterminé par un examen unique de son précédesseur, la préparation complète des ensembles D_i et P_i demande au maximum O(D) opérations. La fabrication d'un tas (heap) comprenant les éléments lexicographiquement minimum de chacun des ensembles D_i demande O(nlogn) opérations. Lors de l'étape 1, il est aussi nécessaire de déterminer, par un examen de tous les arcs, l'arc lexicographiquement minimum à chaque noeud, ce qui demande O(m) opérations. La complexité, en pire cas de l'étape 1, est globalement de O(m+D+nlog n) opérations.

L'utilisation d'un tas (heap) permettant l'accès rapide à l'élément lexicographiquement minimum de chaque ensemble D_i permet de simplifier les étapes 2 et 3. Dans le cas simple, le choix de l'étiquette à remplacer se limite à mettre à jour le tas et à accéder à son sommet.

S'il est possible de retarder le remplacement d'une étiquette, les étiquettes dont le remplacement est reporté sont ajoutées (en ordre lexicographique) à une file d'éléments du seau traité.

Dans ce cas, l'étape 2 consiste à définir les limites du seau en trouvant la plus petite étiquette à remplacer en comparant l'étiquette au

sommet du tas et la première étiquette de la file. L'étape 3 consiste à mettre à jour le tas et à choisir la prochaine étiquette à remplacer, i.e. le sommet du tas, s'il appartient au seau traité ou, sinon, un élément de la file.

Dans les deux cas, ces étapes seront exécutées au maximum D fois. Chaque exécution demande O(log n) opérations pour la mise à jour du tas et le choix de l'étiquette à traiter. Globalement, les étapes 2 et 3 demandent dans le pire cas O(Dlog n) opérations.

La quatrième étape peut être implantée de deux façons, selon la structure de données utilisée.

L'utilisation d'une table de dimension D pour conserver toutes les étiquettes possibles permet un accès direct à chaque étiquette et une vérification facile de la dominance entre deux étiquettes à un noeud. La construction des bornes inférieure et supérieure dans une zone d'incertitude demande le calcul du coût de chacun des chemins en provenance des m prédécesseurs et dont l'étiquette est dans la zone d'incertitude. Ce calcul demande l'accès, en temps constant, aux étiquettes des prédécesseurs. La vérification de la dominance dans une table se limite à vérifier si le coût du chemin est plus petit que le coût actuel. Globalement, chaque noeud ayant au maximum d étiquettes et chaque étiquette étant traitée au maximum une fois si $\alpha=1$ lors du calcul de l'intervalle d'incertitude, l'étape 4 demande pour le remplacement des étiquettes éliminées des n noeuds $O(d \sum_{i=1}^{n} m_i) = O(m\ d)$ opérations où m_i est le nombre d'arcs entrants au noeud i.

L'utilisation d'une liste pointée pour conserver les étiquettes efficaces implique un accès séquentiel aux étiquettes efficaces. L'utilisation d'une table de dimension d pour représenter toutes les étiquettes possibles dans la zone d'incertitude permet de simplifier la vérification

de la dominance et de profiter d'une partie des avantages d'une table, tout en conservant l'économie d'espace des listes pointées. La construction des bornes inférieure et supérieure dans la zone d'incertitude demande le calcul du coût de chacun des chemins en provenance des prédécesseurs et ayant leur étiquette dans la zone d'incertitude. L'accès séquentiel aux étiquettes des précédesseurs demande O(d) opérations. La vérification de la dominance s'effectuant dans la table locale se limite à vérifier si le coût du chemin est plus petit que le coût actuel. Lors de la fabrication des nouveaux ensembles D_j et P_j, il y a vérification de la dominance entre les étiquettes de la table et construction d'une liste pointée des étiquettes efficaces de D_j et P_j. Globalement, chaque noeud ayant au maximum d étiquettes et chacune des ces étiquettes étant traitée au maximum α fois, l'étape 4 demande pour le remplacement des étiquettes éliminées des n noeuds $O(\alpha d^2 \sum_i m_i) = O(\alpha m d^2)$ opérations. Le choix du coefficient α sera discuté dans la prochaine section.

Tableau 5.1

Sommaire de la complexité pour la réoptimisation

implantation	espace-mémoire	temps
avec table	$\Omega(m+D)$	$O(D\log n + \alpha md)$
avec liste pointée	$O(m+D)$	$O(D\log n + \alpha md^2)$

5.7 RÉSULTATS NUMÉRIQUES

L'algorithme de réoptimisation (REOPT), présenté à la section 5.4, a été implanté en FORTRAN 77 en utilisant des listes pointées pour représenter les étiquettes efficaces. REOPT a été testé sur un ordinateur CDC CYBER 173 pour comparer ses performances à celles de l'algorithme AMPG. La comparaison entre les deux algorithmes s'effectuera en variant trois paramètres :

1. Le nombre de noeuds n composant le réseau. Nous résoudrons des problèmes de 500, 1000 et 2500 noeuds.

2. Le nombre moyen d'arcs par noeud. Nous choisirons aléatoirement parmi les arcs réalisables selon (5.1) de façon à obtenir au total environ 10*n, 25*n ou 100*n arcs dans le réseau.

3. La largeur des fenêtres de temps. Les débuts de fenêtres a_i seront distribués entre 0 et 100 selon une loi uniforme et la fenêtre de chaque noeud prendra une largeur de 10, 25, 50 ou 100 unités de temps.

Six problèmes test différents ont été générés pour chaque combinaison réalisable des trois paramètres. Ces problèmes ont été produits en utilisant la même méthode que Desrosiers, Pelletier et Soumis [DESR83].

Les problèmes ont été produits en utilisant les paramètres suivants :

1. les tâches sont dispersées dans un carré de [0,70] x [0,70] selon une loi uniforme;

2. les durées des tâches d_i sont dispersées dans l'intervalle [5,15] selon une loi uniforme;

3. le temps inter-tâches t_{ij} est égal à la distance euclidienne entre les taches i et j additionnée de la durée de la tâche i;

4. le coût inter-tâches c_{ij} est égal à t_{ij} moins une grande constante (33 333). Cette constante représente la valeur marginale d'un noeud lors de la génération de colonnes. La constante choisie rend le coût de presque toutes les routes de trois noeuds ou plus négatif.

L'ordinateur utilisé ne permettait pas de conserver en mémoire centrale la totalité des noeuds, des arcs et des étiquettes. Les noeuds et les étiquettes sont conservés en mémoire centrale, alors que les arcs sont divisés en pages et que l'on ne conserve qu'une seule page en mémoire centrale. Lors d'un défaut de page, i.e. lorsque l'exécution de l'algorithme nécessite des arcs d'une autre page, cette nouvelle page est amenée

en mémoire centrale. Ce processus peut se comparer à celui de la mémoire virtuelle sur d'autres ordinateurs. Le temps de gestion de la mémoire est comptabilisé séparément du temps d'exécution de l'algorithme.

Pour effectuer la comparaison entre les deux algorithmes, les temps d'exécution et de gestion de la mémoire seront utilisés. À titre d'information, le nombre d'étiquettes traitées par AMPG, i.e. celles dont les successeurs sont calculés, et le nombre d'étiquettes remplacées par REOPT sont indiqués dans les tableaux.

De plus, le pourcentage d'étiquettes possibles non-réalisables à remplacer (% inconnu) est indiqué dans le tableau 5.2 et permet de mesurer la quantité de travail à effectuer par l'algorithme de réoptimisation remplaçant les étiquettes éliminées pour obtenir la solution optimale.

Pour les problèmes de réoptimisation du tableau 5.2, ce pourcentage est de 3% à 25% du nombre d'étiquettes possibles, D. Pour les problèmes d'optimisation du tableau 5.3, toutes les étiquettes possibles sont à remplacer. La borne inférieure est initialisée à $-\infty$ et les ensembles D_i, à $\{(a_i,-\infty)\}$. Il s'agit du pire cas possible de réoptimisation. Cette série de problèmes permet de comparer les deux algorithmes pour l'obtention de la première solution. Les problèmes de réoptimisation du tableau 1 sont obtenus en retirant de cette première solution les noeuds appartenant au plus court chemin réalisable. Le nombre de noeuds dans le problème résiduel $|N_R|$ est contenu dans le tableau 5.2.

Dans les tableaux 5.2 et 5.3, $|E|$ traitées représente le nombre d'étiquettes ayant été traitées, i.e. le nombre de marquage des successeurs (reaching) effectués. $|E_r|$ représente le nombre d'étiquettes ayant été remplacées, i.e. le nombre de marquage par les prédecesseurs (pulling) effectués. Nous pouvons constater que $|E_r|$ est toujours beaucoup plus

petit que |E| traitée, i.e. que le nombre de traitement effectué est plus petit avec le nouvel algorithme.

La figure 5.3 illustre l'influence du coefficient α sur le temps d'exécution de l'algorithme REOPT pour un problème. Il s'agit d'un problème d'optimisation de 500 noeuds, 12 423 arcs et avec des fenêtres de temps de 100 unités et du problème de réoptimisation obtenu en retirant les dix noeuds du chemin réalisable le plus court. La meilleure valeur de α pour ce problème est comprise entre 2 et 3.

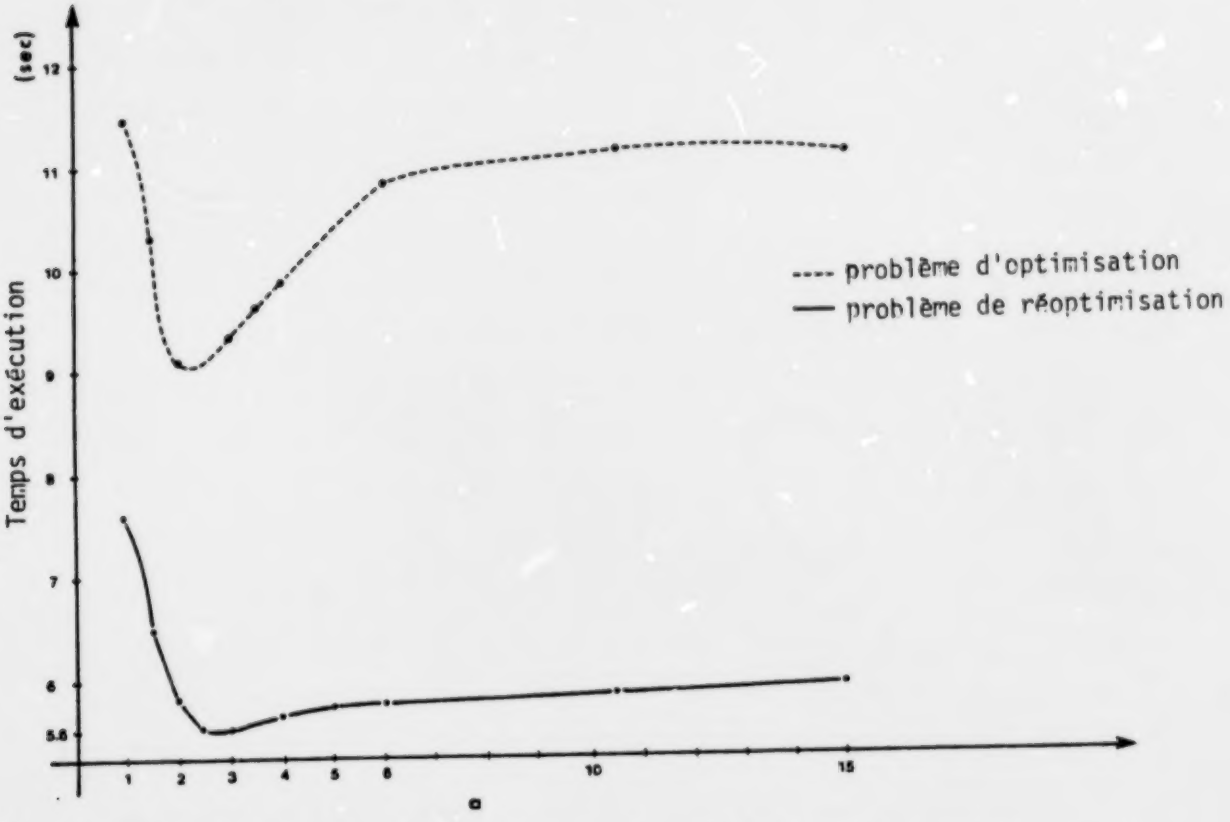


Figure 5.3 : Effet de la taille de la zone d'exploration sur le temps d'exécution de l'algorithme de réoptimisation

La zone d'incertitude est située de $[T,T+\alpha d_j]$ sur l'axe du temps. Le théorème 5.1 nous garantit qu'aucune étiquette duale ne sera située dans $[T,T+d_j]$. Il est probable que le fait qu'il n'y ait aucune étiquette

duale dans l'intervalle $[T,T+\alpha.d_j]$ soit assez fréquent pour réduire le nombre d'étiquettes à remplacer au point de produire une baisse de 25% du temps de calcul entre $\alpha=1$ et $\alpha=2$ pour les deux catégories de problèmes. Pour cette raison, la version de REOPT présentée utilise $\alpha=2$.

La comparaison des temps d'exécution et de gestion de la mémoire pour les problèmes de réoptimisation du tableau 5.2 permet de constater que le temps d'exécution de AMPG est de deux à dix fois plus grand que le temps d'exécution de REOPT; pour la gestion de la mémoire, le temps de AMPG est en moyenne le double de celui de REOPT. L'examen des temps d'exécution par taille de problème montre que, pour les problèmes de 500 noeuds, REOPT est en moyenne 2,2 fois plus rapide que AMPG; pour les problèmes de 1000 noeuds, ce facteur passe à 3,7 et, pour les problèmes de 2500 noeuds, à 5,7. L'utilisation de la réoptimisation permet donc une économie appréciable de temps de calcul.

La comparaison des temps d'exécution et de gestion de mémoire pour les problèmes d'optimisation du tableau 5.3 permet de déterminer si l'algorithme primal-dual REOPT est encore compétitif face à l'algorithme primal AMPG pour la résolution de problèmes complets. L'examen des résultats du tableau 5.3 montre que les temps d'exécution des deux algorithmes sont en moyenne les mêmes avec un très léger avantage d'environ 5% en moyenne pour REOPT. L'algorithme primal-dual de la section 5.4 est donc compétitif tant pour l'optimisation que pour la réoptimisation.

5.8 CONCLUSION

L'algorithme de réoptimisation développé répond aux objectifs de départ. Il permet de résoudre de deux à dix fois plus rapidement les problèmes réduits. La mise au point d'une méthode primale-duale pour résoudre les équations de récurrence du problème de plus court chemin avec contraintes d'horaires présente aussi d'autres intérêts.

TABLEAU 5.2 : Statistiques d'exécution des deux algorithmes pour des problèmes de réoptimisation

					AMPG							REOPT (α=2)				
							Temps							Temps		
|N|				#		Gestion mémoire		Exécution		#			Gestion mémoire		Exécution	
réseau	D	|A|	$|E_s|$	$|N_r|$ problèmes	problèmes résolus	|E| traitées	Moy (sec.)	Max (sec.)	Moy (sec.)	Max (sec.)	problèmes résolus	% inconnu	$|E_r|$	Moy (sec.)	Max (sec.)	Moy (sec.)	Max (sec.)
500	5426	5044	745	493	6	745	–	–	.50	.56	6	.03	25	–	–	.13	.14
500	5412	12487	898	492	6	898	.36	.46	1.10	1.35	6	.05	46	.12	.13	.21	.25
500	5387	24982	1016	490	6	1016	.77	1.01	2.12	2.58	6	.14	99	.29	.39	.51	.94
500	5393	49984	1119	490	6	1119	1.56	2.11	4.31	5.15	6	.15	109	.77	1.09	.97	1.04
500	12809	4993	1214	492	6	1214	–	–	.68	.85	6	.09	100	–	–	.27	.30
500	12783	12527	1646	491	6	1646	.62	.80	1.83	2.26	6	.09	127	.30	.41	.49	.66
500	12748	24907	2089	490	6	2089	1.56	2.40	4.12	5.19	6	.19	239	.76	.88	1.25	1.67
500	12774	49988	2399	491	6	2399	2.91	3.73	8.77	10.01	6	.18	246	1.55	1.76	2.11	3.92
500	25092	5033	2065	492	6	2065	–	–	1.15	1.39	6	.15	291	–	–	.68	.80
500	25126	12501	3128	492	6	3128	1.55	1.83	3.30	3.87	6	.14	309	.68	.81	1.31	1.71
500	24973	24975	4090	489	6	4090	3.33	3.76	8.26	9.66	6	.23	542	1.57	1.71	3.65	4.81
500	25160	49993	4569	493	6	4569	6.32	7.19	18.52	21.88	6	.15	395	2.90	3.54	4.98	6.83
500	49910	4952	4102	494	6	4102	–	–	2.43	2.70	6	.13	405	–	–	1.13	1.39
500	49692	12477	6129	492	6	6131	2.28	2.98	7.71	8.79	6	.21	801	1.34	1.56	4.12	6.24
500	49557	25074	7325	490	6	7325	5.63	6.94	17.97	20.33	6	.22	1033	2.65	3.61	9.56	13.69
500	49523	49913	10177	490	6	10177	9.86	12.69	52.17	58.27	6	.26	1327	4.46	4.88	30.62	39.48
1000	10923	9993	1499	993	6	1499	–	–	1.01	1.13	6	.05	74	–	–	.27	.38
1000	10912	25015	1815	992	6	1815	.85	1.06	2.31	2.71	6	.05	78	.34	.68	.39	.53
1000	10901	50129	2053	991	6	2053	1.97	2.59	4.58	5.52	6	.08	127	.61	.77	.73	1.33
1000	10886	99739	2396	989	6	2396	4.13	5.32	9.94	12.11	6	.09	140	1.67	2.04	1.25	2.04
1000	25809	10018	2428	992	6	2428	–	–	1.41	1.69	6	.05	113	–	–	.39	.51
1000	25757	25058	3538	990	6	3538	1.35	1.80	4.21	5.12	6	.03	74	.60	.85	.47	.64
1000	25714	50147	4236	989	6	4236	2.94	4.18	9.33	10.87	6	.12	333	1.74	1.91	1.89	2.73
1000	25740	100103	4943	990	6	4946	6.98	9.86	21.08	23.92	6	.09	290	3.84	4.16	2.66	3.00
1000	50358	10017	4255	991	6	4255	–	–	2.53	2.93	6	.03	137	–	–	.57	.60
1000	50375	25080	6516	991	6	6516	2.79	3.55	7.80	9.40	6	.12	661	1.52	1.86	2.88	3.94
1000	50456	50016	8203	989	6	8203	6.64	8.58	18.59	21.34	6	.13	712	3.34	3.77	5.32	9.12
1000	50541	99959	10243	991	6	10243	16.40	17.14	44.60	50.90	6	.13	707	4.73	4.94	9.37	12.90
1000	99905	10012	7776	989	6	7776	–	–	4.75	5.54	6	.13	845	–	–	2.37	3.24
1000	99990	25065	13171	990	4	12732	6.05	6.92	16.98	19.67	6	.19	1699	1.98	2.20	8.86	11.37
1000	100394	50051	13910	992	1	13910	12.61	12.61	37.89	37.89	1	.06	3035	4.94	4.94	25.50	25.50
2500	27419	24957	3772	2492	6	3760	2.25	3.08	2.56	2.76	6	.02	67	.80	.94	.46	.52
2500	27408	62446	4566	2491	6	4559	7.03	9.59	6.17	6.53	6	.03	118	3.32	4.57	.74	.84
2500	27401	124902	5173	2491	6	5170	14.18	19.33	12.38	14.30	6	.04	201	6.46	9.18	1.44	1.92
2500	27386	249558	5883	2489	6	5883	32.91	39.78	26.32	28.44	6	.06	291	12.90	16.26	2.86	3.18
2500	64792	25060	6096	2492	6	6074	3.32	3.86	3.57	4.32	6	.02	153	1.64	2.96	.78	1.26
2500	64774	62503	8546	2491	6	8526	12.55	14.31	10.17	11.48	6	.03	247	3.46	5.11	1.41	2.21
2500	64766	125196	10584	2491	6	10571	25.78	30.17	24.02	26.96	6	.08	638	12.53	15.13	3.95	4.91
2500	64705	249806	12649	2488	0	–	–	–	–	–	6	.11	888	24.11	32.62	9.56	13.64

TABLEAU 5.3 - Statistiques d'exécution des deux algorithmes pour des problèmes d'optimisation

				AMPG						REOPT (α=2)					
						Temps mémoire		Exécution				Temps mémoire		Exécution	
|N| réseau	D	|A|	$\|E_s\|$	# problèmes résolus	|E| traitées	Gestion Moy (sec.)	Max (sec.)	Moy (sec.)	Max (sec.)	# problèmes résolus	$\|E_r\|$	Gestion Moy (sec.)	Max (sec.)	Moy (sec.)	Max (sec.)
500	5500	5044	763	6	763	-	-	.50	.57	6	502	-	-	.69	.70
500	5500	12487	932	6	932	.37	.47	1.12	1.37	6	503	.12	.15	1.10	1.14
500	5500	24982	1045	6	1045	.78	1.13	2.21	2.75	6	517	.38	.68	1.81	1.84
500	5500	49984	1160	6	1160	1.59	1.97	4.51	5.45	6	539	.88	1.20	3.16	3.20
500	5500	99984	1231	6	1231	-	-	.70	.83	6	683	-	-	1.10	1.15
500	13000	4995	1231	6	1675	.65	.93	1.87	2.33	6	850	.38	.39	2.14	2.30
500	13000	12527	1675	6	2065	1.59	2.22	4.17	5.17	6	994	.95	1.04	3.91	4.07
500	13000	24907	2065	6	2435	2.88	3.57	9.33	10.82	6	1112	1.95	2.06	7.40	7.60
500	13000	49988	2435	6	2167	-	-	1.22	1.48	6	1130	-	-	2.15	2.24
500	25500	5033	2167	6	3309	1.56	1.94	3.56	4.42	6	1394	.79	.89	4.52	4.84
500	25500	12501	3309	6	4132	3.25	3.63	8.67	10.56	6	1660	1.75	1.83	8.87	9.34
500	25500	24975	4132	6	4843	6.47	7.19	19.65	22.81	6	1944	3.79	3.93	18.04	19.20
500	25500	49953	4843	6	4243	-	-	2.52	3.12	6	1894	-	-	4.19	4.38
500	50500	4952	4243	6	6196	2.33	3.06	7.89	9.35	6	2502	1.64	1.89	9.37	9.72
500	50500	12477	6196	6	7961	5.59	6.60	20.05	24.17	6	3056	2.99	3.48	19.22	20.71
500	50500	25074	7961	6	10408	10.42	13.23	53.71	60.16	6	3551	5.17	5.52	41.16	43.24
500	50500	49913	10408												
1000	11000	9993	1526	6	1526	-	-	1.01	1.14	6	1004	-	-	1.37	1.39
1000	11000	25015	1827	6	1827	.84	1.01	2.33	2.76	6	1023	.41	.70	2.30	2.34
1000	11000	50129	2061	6	2061	1.95	2.43	4.68	5.55	6	1046	1.07	1.37	3.78	3.83
1000	11000	99739	2393	6	2393	4.19	5.17	10.08	11.90	6	1093	2.43	2.58	6.68	6.74
1000	11000	199999	2434	6	2434	-	-	1.43	1.73	6	1362	-	-	2.20	2.33
1000	26000	10018	2434	6	3575	1.34	1.78	4.33	5.24	6	1745	.94	.96	4.56	4.76
1000	26000	25058	3575	6	4278	2.88	3.93	9.63	10.97	6	2022	2.17	2.35	8.22	8.33
1000	26000	50147	4278	6	5077	6.88	9.20	21.84	24.85	6	2240	4.75	4.98	15.43	16.08
1000	26000	100103	5077	6	4314	-	-	2.59	2.96	6	2254	-	-	4.37	4.51
1000	51000	10017	4314	6	6523	2.78	3.58	8.05	9.30	6	2873	1.95	2.12	9.43	9.83
1000	51000	25080	6523	6	8287	6.75	8.79	19.10	22.53	6	3395	4.19	4.31	18.30	19.47
1000	51000	50016	8287	6	10322	16.58	16.97	45.75	53.62	6	3892	6.22	6.35	37.40	39.95
1000	51000	99959	10322	6	8242	-	-	5.03	5.91	6	3769	-	-	8.54	9.01
1000	101000	10012	8242	6	12557	5.67	7.49	17.33	19.72	6	5130	2.37	2.52	19.61	20.13
1000	101000	25063	12830	5	13699	11.98	11.98	37.79	37.79	1	5993	5.24	5.24	35.96	35.96
1000	101000	50051	13699	1											
2500	27500	24957	3790	6	3778	2.22	3.00	2.61	2.88	6	2535	2.05	2.78	3.86	3.91
2500	27500	62446	4587	6	4580	7.22	9.77	6.08	6.87	6	2571	5.99	7.93	6.28	6.37
2500	27500	124902	5181	6	5179	13.99	19.12	12.70	14.93	6	2645	12.57	18.44	10.10	10.25
2500	27500	249358	5946	6	5943	32.79	39.87	26.57	28.57	6	2741	25.37	35.90	17.81	18.01
2500	27500	499999	6120	6	6098	3.36	4.30	3.57	4.23	6	3564	3.11	3.79	6.19	6.42
2500	65000	25060	8590	6	8570	12.44	13.93	10.36	11.49	6	4409	8.44	10.31	12.11	12.53
2500	65000	62503	10546	6	10498	25.70	30.14	24.36	27.86	6	5056	18.75	25.04	21.54	21.96
2500	65000	125196	12610	0	-	-	-	-	-	6	5527	36.30	45.46	39.87	40.50
2500	65000	249806													

La nouvelle méthode permet aussi de résoudre des PPCCCH en un temps comparable aux autres méthodes sur des problèmes d'une taille considérable. En effet, il faut environ une minute de temps CPU sur un ordinateur CDC CYBER 173 pour résoudre un problème de 2500 noeuds et 250 000 arcs. De plus, cette méthode peut être utilisée pour réoptimiser des problèmes où les coûts sont augmentés. Lors de la génération de colonnes, si seulement quelques valeurs marginales ont été diminuées d'une itération à l'autre, il devient intéressant d'augmenter les coûts des arcs et de réoptimiser.

Cette nouvelle méthode primale-duale utilise le "pulling" pour modifier localement la solution, ce qui permet de bénéficier à la fois des avantages d'économie de mémoire pour les noeuds non-modifiés et des avantages offerts par l'accès direct aux étiquettes du noeud modifié. Cette nouvelle méthode ouvre la voie pour la mise au point d'algorithmes pour des problèmes de programmation dynamique à plusieurs dimensions.

CHAPITRE VI

UN ALGORITHME POUR LE PROBLÈME DU PLUS COURT CHEMIN AVEC CONTRAINTES SUPPLÉMENTAIRES

Un des sous-problèmes les plus courants lors de la résolution de problèmes de routing, d'ordonnancement et de télécommunications est le problème de plus court chemin. Dans certains modèles, des contraintes supplémentaires sont imposées au sous-problème. L'algorithme décrit dans ce chapitre a été mis au point pour la résolution des sous-problèmes lors de la fabrication d'horaires de travail pour les conducteurs d'autobus urbain par une méthode de génération de colonnes.

Ce chapitre présente une généralisation à plusieurs dimensions de l'algorithme présenté pour la réoptimisation au chapitre précédent. Tout au long du chapitre, nous adapterons les définitions et concepts au cas multidimensionnel et nous donnerons des preuves adaptées pour la version multidimensionnelles des théorèmes.

6.1 LE PROBLÈME

Soit le réseau G=(N,A) ou A est l'ensemble des arcs et N est l'ensemble de nœuds dont une source p et un puits q. Le coût $c_{ij} \in R$ positif ou négatif, associé à chaque arc $(i,j) \in A$ permet de définir le problème de plus court chemin (6.1-6.3).

$$\text{Minimiser} \quad c.x = \sum_{(i,j)\in A} c_{ij}\, x_{ij} \tag{6.1}$$

sujet à

$$x \in X(G) = \begin{cases} \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = \begin{cases} +1, & i=p \\ 0, & i\in N,\ i\neq p,\ i\neq q \\ -1, & i=q \end{cases} & (6.2) \\ x_{ij} \geq 0 \qquad \text{pour tout } (i,j)\in A & (6.3) \end{cases}$$

Dans notre problème, nous avons des contraintes sur l'utilisation de L ressources. À chaque arc (i,j) et à chaque ressource ℓ, est associée $d_{ij}^{\ell} \in N$, la quantité de ressource ℓ utilisée par l'arc (i,j). Lorsque les noeuds sont des tâches, le coût et les ressources utilisées par un noeud i sont ajoutés à ceux des arcs extrants (i,j). Soit T_j^{ℓ}, la quantité de ressource ℓ utilisée en poursuivant un chemin depuis la source jusqu'au noeud j; notre problème consiste à trouver le chemin élémentaire de coût minimum entre p et q tel que T_j^{ℓ} respecte la fenêtre de réalisabilité $[a_j^{\ell}, b_j^{\ell}]$ pour tous les noeuds du chemin et pour toutes les ressources $\ell=1,\ldots,L$. Nous considérons le problème où il est possible d'atteindre un noeud en utilisant moins d'une ressource ℓ que le minimum de la fenêtre de réalisabilité de cette ressource. En gaspillant la ressource ℓ, il est alors possible de débuter la visite de ce noeud au minimum de la fenêtre de réalisatilité de cette ressource. Le problème du plus court chemin avec fenêtre de temps [DESR 83] en est un exemple; il est possible d'arriver à un noeud avant le début de la fenêtre de temps et d'attendre, i.e. gaspiller du temps, avant de débuter la visite du noeud au début de la fenêtre de temps.

Pour obtenir le problème de plus court chemin avec contraintes supplémentaires (PPCCCS), nous devons ajouter au problème de plus court chemin (6.1-6.3) les contraintes suivantes :

$$x_{ij} > 0 \rightarrow T_i^{\ell} + d_{ij}^{\ell} \leq T_j^{\ell} \qquad (i,j)\in A,\ \ell=1,..,L \qquad (6.4)$$

$$a_i^{\ell} \leq T_i^{\ell} \leq b_i^{\ell} \qquad i\in N,\ \ell=1,\ldots,L \qquad (6.5)$$

où x_{ij} représente le flot sur l'arc (i,j) et T_i^{ℓ}, la quantité de la $\ell^{ième}$ ressource utilisée pour parvenir au noeud i.

Si le gaspillage de ressource n'est pas permis, il faut remplacer l'inégalité de l'équation (6.4) par une égalité. Le problème résultant est plus contraint et sa solution optimale peut posséder un coût plus élevé. Ce modèle correspond à la programmation dynamique classique (voir Jaffe [JAFF84]) et est généralement plus complexe à résoudre que si le gaspillage est permis.

Nous considérons que tous les arcs non-réalisables sont exclus de l'ensemble A. Un arc (i,j) est non-réalisable s'il ne permet pas de passer du noeud i au noeud j en respectant les fenêtres de réalisabilité en i et j. Tous les arcs $(i,j)\in A$ respectent donc la condition suivante pour toutes les ressources :

$$a_i^\ell + d_{ij}^\ell < b_j^\ell \qquad \ell=1,\ldots,L \qquad (6.6)$$

La fabrication d'horaires de travail pour les chauffeurs d'autobus urbains consiste à déterminer le nombre et la composition des journées de travail permettant de réaliser tous les noeuds de façon à minimiser le coût total. Nous utilisons une méthode de génération de colonnes, similaire à celle utilisée par Desrosiers, Soumis et Desrochers [DESR84a] pour résoudre le problème de fabrication de routes d'autobus scolaires.

Cette méthode consiste à construire des journées de travail réalisables jusqu'à l'obtention d'une solution recouvrant de façon optimale l'ensemble des noeuds. À chaque itération de la méthode, nous cherchons à améliorer la solution actuelle en insérant dans la base, la journée de coût marginal minimum obtenue en résolvant un problème de plus court chemin avec contraintes.

La différence entre ces applications se situe au niveau du sous-problème à résoudre. Dans le cas des routes d'autobus scolaires, nous devions résoudre un problème de plus court chemin avec une contrainte supplémentaire [DESR83], dans le cas des journées de travail, le sous-problème comporte plusieurs contraintes supplémentaires. Ces contraintes supplémentaires portent sur la durée totale d'une journée, le nombre de noeuds dans une journée, sa durée utile,...

6.2 REVUE DE LA LITTÉRATURE

Bien que le problème du plus court chemin avec contraintes supplémentaires et le problème voisin du plus court chemin multicritère, aient

fait l'objet de près d'une quinzaine d'articles, nous ne discuterons ici que de ceux utilisant notre approche : la programmation dynamique. Pour une revue de littérature plus complète, consulter le chapitre 4.

Introduisons d'abord une définition des états commune à la plupart des approches par programmation dynamique pour ces problèmes. À chaque chemin X_{pj} de l'origine à un noeud j, est associée une étiquette (quantité de ressources utilisées, coût) correspondant aux L quantités de ressources utilisées T_j^ℓ et au coût du chemin X_{pj}. Ces étiquettes sont représentées par $(T_j^{1k}, T_j^{2k}, \ldots, T_j^{Lk}, C_j^k)$ et dénotent les caractéristiques du $k^{ième}$ chemin X_{pj}^k, les indices k pourront être absents si le contexte n'est pas ambigu. De plus, $T_j^{\cdot k}$ ou $T_j^{\cdot}$ seront utilisés pour représenter le vecteur de dimension L. Pour le traitement de ces problèmes par programmation dynamique, l'ensemble des états est $\{T_j^{\cdot k} \mid \text{pour tous k et j}\}$ et C_j^k est le coût associé à l'état $T_j^{\cdot k}$. Un chemin X_{pj}^k et l'état associé $T_j^{\cdot k}$ sont réalisables si $a_j^\ell \leq T_j^{\ell k} \leq b_j^\ell$ pour toutes les ressources $\ell=1,\ldots,L$. Dans le cas des problèmes de plus court chemin multicritère, la définition des états est la même mais l'algorithme optimise toutes les composantes de l'état et non plus uniquement le coût.

Marquage des successeurs (reaching)

Il existe deux méthodes de ce type. La première méthode, basée sur la généralisation de l'algorithme de Ford-Bellman [JOKS66, VINC74, HABE81, DESR 83, JAFF84], fonctionne par correction des étiquettes. La seconde méthode, basée sur la généralisation de l'algorithme de Dijkstra [YACU 73, MINO75, HANS80, ANEJ83], fonctionne par marquage permanent des étiquettes. L'étape de base dans les deux algorithmes, le reaching, consiste, à partir d'une étiquette associée à un chemin X_{pi}, à marquer les étiquettes correspondant aux chemins réalisables X_{pj} obtenus en prolongeant le chemin X_{pi} par tous les arcs $(i,j) \in A$. Ces algorithmes créent pour chaque noeud i de nouvelles étiquettes à plusieurs noeuds j; ainsi des étiquettes sont créées

plusieurs reprises à chaque noeud j si le noeud j possède plusieurs prédécesseurs. Il faut donc effectuer plusieurs mises à jour des étiquettes à chaque noeud j. Cette mise à jour est coûteuse surtout s'il y a plusieurs contraintes supplémentaires.

Marquage par les prédécesseurs (pulling)

Cette méthode primale-duale, mise au point pour la réoptimisation du problème du plus court chemin avec fenêtre de temps, calcule les chemins X_{pj} associés à une partie ou à tous les états d'un noeud j. Pour un ensemble S d'états du noeud j, le pulling consiste à marquer les étiquettes ($T_j^{\cdot}$ = (max $\{a_j^{\ell}, T_i^{\ell k}+d_{ij}^{\ell}\}$, $\ell=1,\ldots,L$), $C_i^k+c_{ij}$) associées aux chemins X_{pj} obtenus en prolongeant tous les chemins X_{pi} auxquels l'ajout de l'arc $(i,j)\in A$ permet l'arrivée au noeud j en un état $T_j^{\cdot}\in S$. Les nouvelles étiquettes étant créées à un seul noeud, cet algorithme demande, au contraire des algorithmes basés sur le reaching, la mise à jour des étiquettes à un seul noeud.

6.3 LES PRINCIPES DU PULLING MULTIDIMENTIONNEL

Les principes du pulling ont été développés au chapitre précédent pour le cas d'une contrainte supplémentaire. Il est possible de les généraliser pour les appliquer au problème du plus court chemin avec plusieurs contraintes supplémentaires.

Définition 6.1

Soit X^1 et X^2 deux chemins distincts de p à j et leurs étiquettes associées $(T_j^{\cdot 1},C_j^1)$ et $(T_j^{\cdot 2},C_j^2)$, X^1 domine X^2 ou $(T_j^{\cdot 1},C_j^1) < (T_j^{\cdot 2},C_j^2)$ si et seulement si $C_j^1 \leq C_j^2$, $T_j^{\ell 1} \leq T_j^{\ell 2}$ pour tout $\ell=1,\ldots,L$ et $(T_j^{\cdot 1},C_j^1) \neq (T_j^{\cdot 2},C_j^2)$.

La relation de dominance nous permet de conclure qu'un chemin X_{pj} et son étiquette associée (T_j^*, C_j) sont non dominés (efficaces) si et seulement si X_{pj} est le chemin de coût minimal arrivant au noeud j avec des valeurs inférieures ou égales à T_j^ℓ (ℓ=1,...,L).

Nous utilisons le principe d'optimalité de Bellman, i.e. que tout sous-chemin d'un chemin efficace doit lui aussi être efficace, pour définir les équations de récurrence suivantes qui caractérisent les chemins efficaces de la source à tous les noeuds pour le PPCCCS :

Pour tout $j \in N$ et T_j^* réalisable :

$$C(j,T_j^*) = \min_{\substack{i\in N,\ T_j^* \text{ état réalisable}}} \{C(i,T_i^*)+c_{ij} \mid T_i^\ell + d_{ij}^\ell \leq T_j^\ell, (i,j)\in A. \quad \ell=1,\ldots,L\}$$

$$C(p,0^*) = 0$$

L'application du principe d'optimalité permet d'accélérer le calcul des équations de récurrence en évitant de prolonger des chemins X_{pi}^k dominés et de produire ainsi de nouveaux chemins X_{pj} dominés.

La relation de dominance n'étant pas une relation d'ordre total, elle ne permet pas d'ordonner tous les chemins. Ceci implique la possibilité de plusieurs chemins efficaces pour chaque noeud. Nous dénotons par $EFF\{R_j\}$, les étiquettes efficaces (non-dominées) d'un ensemble d'étiquettes R_j. La figure 6.1 illustre pour un noeud la relation entre différentes étiquettes (valeur, coût) associées à des chemins efficaces (X^1 à X^4) et à d'autres dominés (X^5 à X^8) pour le cas d'une seule contrainte supplémentaire.

La relation de dominance permet de définir le coût du chemin de coût minimal en fonction de l'état. Tout ensemble d'étiquettes $R_j = \{(T_j^{*k}, C_j^k)\}$ associées à un noeud j permet de définir une telle fonction. La figure 6.1 illustre cette fonction pour un cas simple.

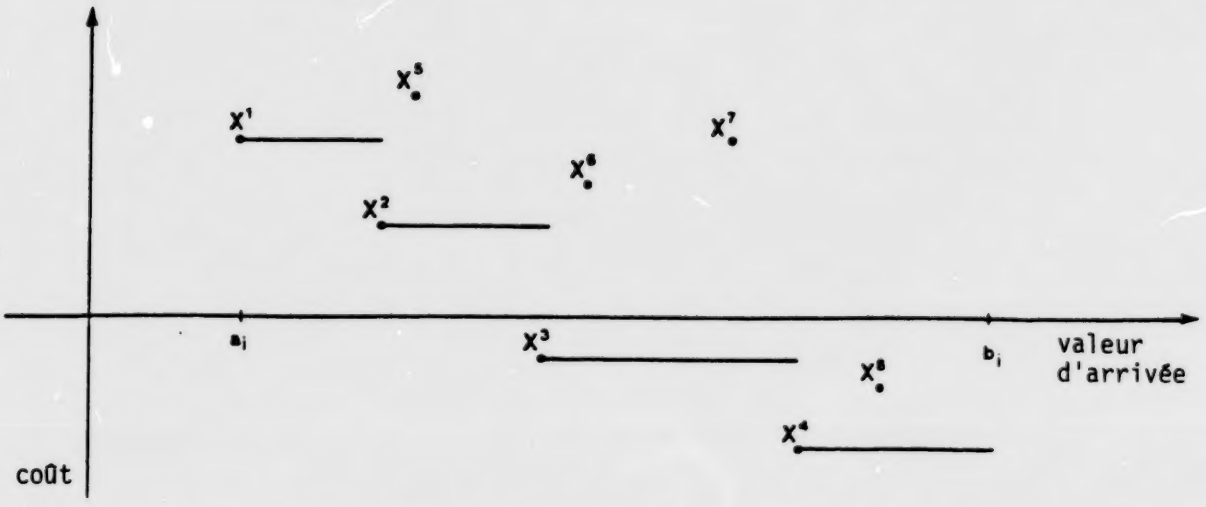


Figure 6.1 : Relation de dominance entre étiquettes associées à différents chemins

Soit R_j un ensemble d'étiquettes associées à des chemins X^k_{pj}, pour tous les états T^*_j du noeud j, la fonction $F_R(T^*_j)$ est définie par

$$F_R(T^*_j) = \min \{C^k_j \mid T^{\ell k}_j < T^\ell_j, \quad \ell=1,\ldots,L, \text{ pour } (T^{*k}_j, C^k_j) \in R_j\}.$$

La fonction $F_R(T^*_j)$ représente le coût minimum pour un chemin X_{pj} ayant utilisé moins de T^*_j unités des ressources.

L'algorithme demande la création de deux ensembles d'étiquettes à chaque noeud. Chaque ensemble d'étiquettes définit une fonction. L'algorithme modifie les ensembles d'étiquettes, modifiant ainsi les fonctions associées, jusqu'à l'obtention de la solution. Le premier ensemble P_j comprend des étiquettes associées à des chemins réalisables et définit la fonction primale. La fonction primale est une borne supérieure sur le coût des solutions efficaces à chaque noeud. Si P_j est vide, la valeur de la fonction primale est définie comme infiniment grande. Le second ensemble Q_j comprend des étiquettes associées à des chemins réalisables ou non. Les étiquettes de Q_j ne doivent pas être dominées par des étiquettes associées à des chemins réalisables. Les étiquettes de Q_j définissent la

fonction duale qui est une borne inférieure sur le coût des solutions efficaces à chaque noeud.

Il est possible d'initialiser P_j et Q_j de deux façons. Si nous ne disposons pas d'informations, $P_j=\emptyset$ et une étiquette $(a_j^1,\ldots,a_j^L,-\infty)$ initialise Q_j. En cas de réoptimisation après une augmentation des coûts, Q_j sera initialisé avec toutes les vieilles étiquettes car elles demeurent des bornes inférieures tandis que P_j ne contiendra que les étiquettes associées à des chemins toujours réalisables car ces chemins demeurent optimaux.

Lorsque les fonctions primale et duale ont la même valeur, l'étiquette de P_j et Q_j à cet état est associée à un chemin efficace pour cet état. Si les fonctions primale et duale à un état ont des valeurs différentes, nous sommes dans l'incertitude face au coût du chemin efficace en cet état. Une zone d'incertitude est un état, ou un ensemble d'états, où les fonctions primale et duale n'ont pas la même valeur et dont la valeur de la fonction duale est définie par la même étiquette. Une zone d'incertitude est caractérisée par l'étiquette de Q_j-P_j qui domine les autres étiquettes associées à la zone d'incertitude.

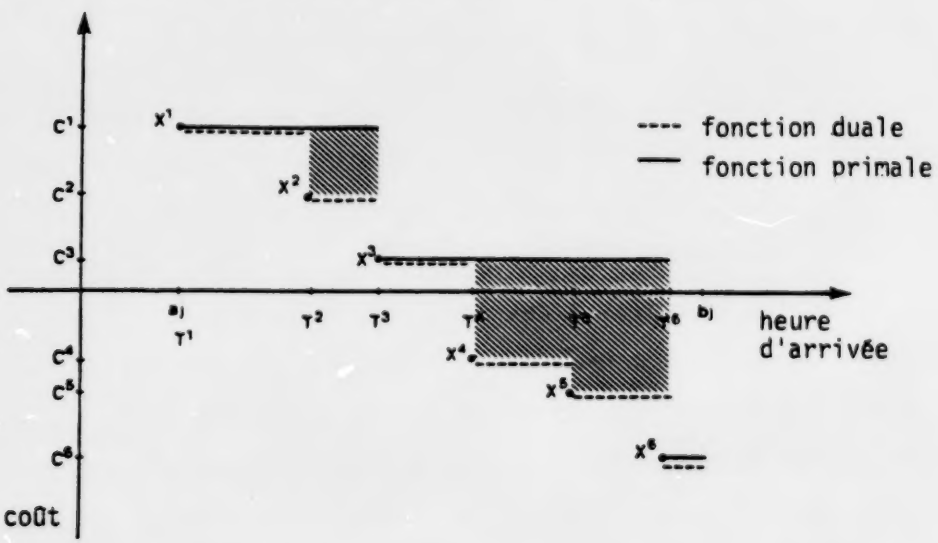


Figure 6.2 - Exemple de fonctions primale et duale

La figure 6.2 illustre les fonctions primale et duale pour l'ensemble $P_j = \{(X^1,C^1),(X^3,C^3),(X^6,C^6)\}$ et l'ensemble $Q_j = \{(X^1,C^1),(X^2,C^2),(X^3,C^3),(X^4,C^4),(X^5,C^5),(X^6,C^6)\}$. Les états de la zone hachurée font partie de la zone d'incertitude.

Au chapitre précédent, nous avons défini la zone d'incertitude par les étiquettes. Nous préférons ici définir la zone d'incertitude comme les états où les fonctions primale et duale sont différentes.

Il y a deux façons de réduire la taille d'une zone d'incertitude. La première est d'augmenter la valeur de la fonction duale. La seconde façon est de diminuer la valeur de la fonction primale. L'exemple numérique de la section 6.6 illustre ce processus en détail.

La première façon de réduire l'incertitude est de calculer une nouvelle fonction duale pour la zone délimitée. Cette nouvelle borne inférieure est calculée en trouvant tous les chemins X_{pi} dont le prolongement par un arc $(i,j)\in A$ produit un chemin X_{pj} ayant l'état de son étiquette dans la zone délimitée. Le sous-ensemble R_j des étiquettes efficaces est calculé pour obtenir les étiquettes définissant la borne inférieure

$$R_j = EFF\{(T_j^*,C_j) = ((\max\{a_j^\ell, T_i^{*k}+d_{ij}^\ell\},\ \ell=1,\ldots,L),\ C_i^k+c_{ij}) \mid \text{pour tout } (i,j)\in A,\ (T_i^{*k},C_i^k)\in Q_i \text{ et } T_j^* \text{ appartient à la zone à explorer}\}.$$

L'autre façon de réduire l'incertitude est d'améliorer la fonction primale par le calcul de solutions réalisables situées dans la zone délimitée. Il n'y a présentement aucune solution réalisable dans la zone d'incertitude. Les solutions réalisables sont calculées en trouvant tous les chemins réalisables X_{pi} dont le prolongement par un arc $(i,j)\in A$ produit un chemin X_{pj} ayant l'état de son étiquette dans la zone délimitée. De façon à calculer

$$RP_j = EFF\{(T_j^*,C_j) = ((\max\{a_j^\ell, T_i^{*k}+d_{ij}^\ell\},\ \ell=1,\ldots,L),\ C_i^k+c_{ij}) \mid \text{pour tout } (i,j)\in A,\ (T_i^{*k},C_i^k)\in P_i \text{ et } T_j^* \text{ appartient à la zone à explorer}\}$$

l'ensemble des solutions efficaces réalisables formant la nouvelle fonction primale dans la zone délimitée. Certaines des étiquettes de RP_j peuvent être dominées par des étiquettes non-réalisables de R_j.

Le remplacement de l'étiquette éliminée s'effectue

1) en trouvant les éventuelles nouvelles étiquettes réalisables (εRP_j) et efficaces (εR_j) pour les ajouter à P_j, Cet ajout d'étiquettes, modifie à la fois les ensembles définissant la borne supérieure et la borne inférieure, détruit l'incertitude en ce point :

$$P_j \leftarrow P_j \cup (R_j \cap RP_j)$$

Cette règle de remplacement permet de garantir que P_j ne contiendra que des solutions efficaces.

2) en ajoutant les étiquettes efficaces (εR_j) à l'ensemble Q_j après avoir éliminé l'étiquette remplacée

$$Q_j \leftarrow \{Q_j-(T_j^k,C_j^k)\} \cup R_j$$

pour modifier la borne inférieure.

6.4 ORDRE DE TRAITEMENT DES ÉTIQUETTES

Lors du remplacement d'une étiquette éliminée, il faut s'assurer que l'étiquette à remplacer ne soit pas produite à nouveau, entraînant une étape sans réduction de la taille de la zone d'incertitude. Dans [DESR85a], un ordre de traitement et des conditions de qualification sur les arcs sont définis pour la résolution du PPCCCS pour le cas particulier à une contrainte par un algorithme primal. Les mêmes notions sont reprises ici pour définir un ordre de remplacement des étiquettes de $S = \sum_{j\in N} (Q_j-P_j)$ définissant un algorithme primal-dual pour le PPCCCS.

Définition 6.2.

Soit $a,b\in \mathbb{R}^L$, a est lexicographiquement plus petit que b ($a \overset{L}{<} b$) si et seulement si $a\neq b$ et que le premier élément non-nul de b-a est positif.

Définition 6.3.

Un arc (i,j) est lexicographiquement positif si et seulement si $(0,0...0) \overset{L}{<} (d^{\bullet}_{ij}, c_{ij})$.

L'ordre lexicographique est un ordre total dans $\mathbb{R}^L$ compatible avec l'ordre partiel de dominance.

Le concept de seau généralisé multidimensionnel

Un seau B(S) est un ensemble d'étiquettes de S qu'il est possible de remplacer en diminuant la zone d'incertitude. Lors de la définition formelle du seau, deux propriétés sont importantes :

1) toutes les étiquettes de B(S) doivent être prêtes à être remplacées.
2) Savoir s'il est possible de retarder le remplacement d'une étiquette de B(S).

Cette dernière information permet, dans le cas d'un grand réseau où les données relatives à une partie seulement du réseau sont disponibles, de retarder au maximum le traitement des parties non-disponibles du réseau.

La définition du seau généralisé défini au chapitre 4 permet de répondre à ces questions.

Soit $(d^{\bullet}_{.j} c_{.j}) = \underset{(i,j)\in A}{\text{minlex}} \{(d^{\bullet}_{ij}, c_{ij})\}$, la largeur du seau.

Soit $F(S) = \underset{(T^{\bullet}_i, C^k_i)\in S}{\text{minlex}} \{(T^{\bullet k}_i, C^k_i)\}$, l'étiquette à remplacer.

Nous pouvons définir le seau B(S), un ensemble d'étiquettes à remplacer :

$$B(S) = \{T^{\bullet k}_i, C^k_i)\in S \; F(S) \overset{L}{<} (T^{\bullet k}_i, C^k_i) \overset{L}{<} F(S)+(d^{\bullet}_{.i}, c_{.i})\}$$

Théorème 6.1.

Si $(d^{\bullet}_{.i}, c_{.i}) \overset{L}{>} (0,\ldots,0)$ pour tout $i\in N$, alors, lors du remplacement d'une étiquette de B(S),

1) il est impossible de reproduire l'étiquette remplacée;

2) la zone d'incertitude diminue.

Preuve.

1) Supposons qu'il soit possible de reproduire l'étiquette à remplacer $(T_j^{\cdot},C_j)\in B(S)$, il existe donc une étiquette $(T_i^{\cdot},C_i)\in S$ telle que

$$(T_i^{\cdot},C_i) + (d_{ij}^{\cdot},c_{ij}) = (T_j,C_j) \overset{L}{<} F(S) + (d_{.j}^{\cdot},c_{.j})$$

$$\rightarrow (T_i^{\cdot},C_i) + (d_{ij}^{\cdot},c_{ij}) \overset{L}{<} F(S) + (d_{.j}^{\cdot},c_{.j})$$

$$\rightarrow (T_i^{\cdot},C_i) \overset{L}{<} F(S) \quad \text{car} \quad (d_{ij}^{\cdot},c_{ij}) \overset{L}{>} (d_{.j}^{\cdot},c_{.j}) \quad \text{par définition.}$$

Ce qui contredit la définition de F(S).

2) L'algorithme calcule une nouvelle fonction duale en calculant les nouveaux éléments réalisables et non-réalisables de la borne inférieure pour la zone délimitée de la façon suivante :

$R_j = EFF\{(T_j^{\cdot},C_j) = ((\max\{a_j^{\ell}, T_i^{\ell k}+d_{ij}^{\ell}\}, \ell=1,\ldots,L), C_i^k+c_{ij}) \mid (T_i^{\cdot k},C_i^k)\in Q_i,\ (i,j)\notin A$ et $T_j^{\cdot}$ fait partie de la zone délimitée.$\}$

Les étiquettes réalisables de R_j éliminant l'incertitude en définissant à la fois la fonction duale et la fonction primale, seules les étiquettes non-réalisables de R_j, i.e. $\in RP_j$ délimitent l'incertitude due à un écart entre les deux fonctions. Une borne inférieure lexicographique sur la valeur de la plus petite étiquette non-réalisable de R_j est :

$$((\max\{a_j^{\ell}, T_j^{\ell}+d_{.j}^{\ell}\},\ \ell=1,\ldots,L),\ C_j+c_{.j}).$$

En effet, $((\max\{a_j^{\ell}, T_j+d_{.j}^{\ell}\},\ \ell=1,\ldots,L), C_j+c_{.j}) \overset{L}{\leq}$

$$((\max\{a_i^{\ell}, T_i^{\ell k}+d_{ij}^{\ell}\}\ \ell=1,\ldots,L),\ C_i^k+c_{ij})$$

pour tout $(T_i^{\cdot k},C_i^k)\in S$ et $(i,j)\in A$.

Cette borne inférieure sur le coût du premier chemin non-réalisable introduit lors du remplacement de $(T_j^{\cdot k},C_j^k)\in B(S)$ nous garantit que pour tous

les états T tels que

$$T_j \overset{L}{<} T \overset{L}{<} (\max\{a_j^\ell, T_j^\ell + d_{\cdot j}^\ell\},\ \ell=1,\ldots,L)$$

les nouvelles étiquettes calculées font partie de R_j et RP_j, i.e. que pour ces états la fonction primale a la même valeur que la fonction duale entraînant une réduction de la taille de la zone d'incertitude originale.

□

6.5 L'ALGORITHME

L'utilisation du théorème 6.1 permet la mise au point de l'algorithme suivant pour la résolution du PPCCCS.

Étape 1. Initialisation

- préparation des ensembles P_j et Q_j.
- calcul des $(d_{\cdot j}, c_{\cdot j})$ pour chaque noeud.

Étape 2. Calcul des limites du seau

Trouver l'étiquette $(T_j^{\cdot k}, C_j^k)$ de coût lexicographique minimum parmi l'ensemble $S = \bigcup_{j \in N} (Q_j - P_j)$. Si $S = \emptyset$, terminer.

Étape 3. Recherche d'une étiquette $(T_j^{\cdot k}, C_j^k)$ à traiter

- Trouver une étiquette appartenant au premier seau.
- Si le seau est vide, retourner à l'étape 2.

Étape 4. Choix de la zone à explorer parmi la zone d'incertitude

Étape 5. Remplacement de l'étiquette $(T_j^{\cdot k}, C_j^k)$

A) Calcul de la nouvelle fonction duale dans la zone à explorer

$$R_j = EFF\{(T_j^\cdot, C_j) = ((\max\{a_j^\ell, T_i^{\ell k} + d_{ij}^\ell\},\ \ell=1,\ldots,L),\ C_i^k + c_{ij}) \mid$$
$$\text{pour tout } (i,j) \in A,\ (T_i^{\cdot k}, C_i^k) \in Q_i$$
$$\text{et } T_j^\cdot \text{ est dans la zone à explorer.}\}$$

B) Calcul de la fonction primale dans la zone à explorer

$$RP_j = EFF\{(T_j^*, C_j) = ((\max\{a_j^\ell, T_i^{\ell k} + d_{ij}^\ell\}, \ell = 1, \ldots, L), C_i^k + c_{ij}) |$$

pour tout $(i,j) \in A$, $(T_i^{*k}, C_i^k) \in P_i$

et T_j^* est dans la zone à explorer.$\}$

C) Création des nouveaux ensembles Q_j et P_j

- $P_j = P_j \cup (R_j \cap RP_j)$
- $Q_j = (Q_j - \{(T_j^k, C_j^k)\}) \cup R_j$

retourner à l'étape 3.

Seule l'étape 4, le choix de la zone à explorer dans la zone d'incertitude, reste à préciser. Dans l'exemple numérique, toute la zone d'incertitude sera explorée.

6.6 EXEMPLE NUMÉRIQUE

L'exemple numérique suivant permet d'illustrer la résolution d'un problème de plus court chemin avec une contrainte supplémentaire. Il s'agit d'un problème de 4 noeuds, 5 arcs et 22 états illustré à la figure 6.3. Le noeud P, la source, est initialisé à zéro. Les ensembles P_j sont vides pour les autres noeuds, l'ensemble Q_j est initialisé à zéro pour la source et avec l'étiquette $(a_j, -\infty)$ pour les autres noeuds. Ces initialisations nous permettent de calculer les fonctions primale et duale et de déterminer les zones d'incertitude telles qu'illustrées à la figure 6.3.

Pour simplifier l'illustration de l'algorithme, l'exemple utilisé comprend une seule contrainte. Pour simplifier davantage l'exemple, nous fusionnons les étapes 2 et 3 en choisissant toujours le plus petit élément du seau B(S) éliminant le besoin de calculer les limites du seau à chaque itération.

À la première itération, l'ensemble S des étiquettes à remplacer comprend les étiquettes $(1,-\infty)$ au noeud B, $(3,-\infty)$ au noeud C et $(5,-\infty)$ au

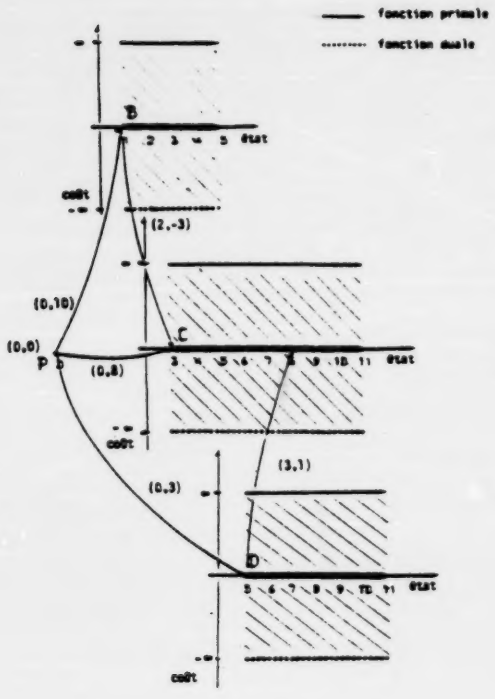


Figure 6.3 - Initialisation et réseau

noeud D. Nous choisissons de traiter l'étiquette $(1,-\infty)$ au noeud B. La zone d'incertitude comprend tous les états de ce noeud. La source est le seul prédécesseur et nous obtenons l'étiquette (1,10) associée à un chemin réalisable, les ensembles $Q_B = P_B = \{(1,10)\}$ et l'élimination de l'incertitude à ce noeud. Le résultat est illustré par la figure 6.4.

Au début de la seconde itération, l'ensemble S comprend $\{(3,-\infty), (5,-\infty)\}$. Nous choisissons de chercher les prédécesseurs de l'étiquette $(3,-\infty)$ au noeud C. La zone d'incertitude comprend tous les états du noeud C. Ce noeud a trois prédécesseurs P, B et D. Nous obtenons les étiquettes $\{(3,8), (3,7), (8,-\infty)\}$ et les ensembles $Q_C = \{(3,7), (8,-\infty)\}$ et $P_C = \{(3,7)\}$. L'incertitude a été éliminée aux états associés aux valeurs 3 à 7. Le résultat est illustré par la figure 6.5.

Au début de la troisième itération, l'ensemble S comprend $\{(5,-\infty), (8,-\infty)\}$. Nous choisissons de traiter l'étiquette $(5,-\infty)$ du noeud D. La

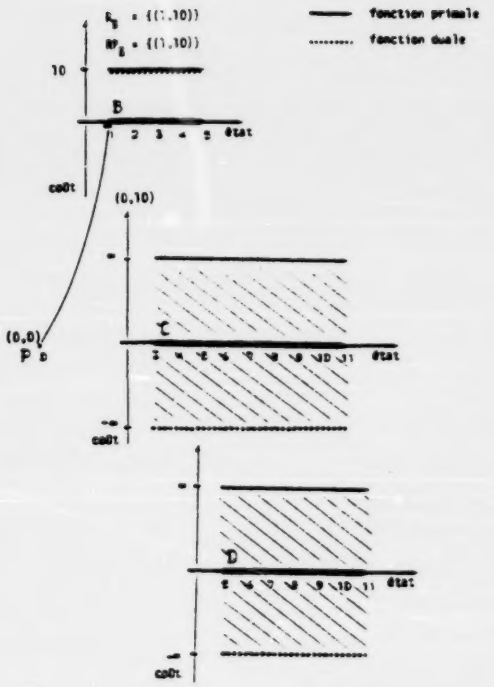


Figure 6.4
Pulling de l'étiquette (1,-∞)
au noeud B

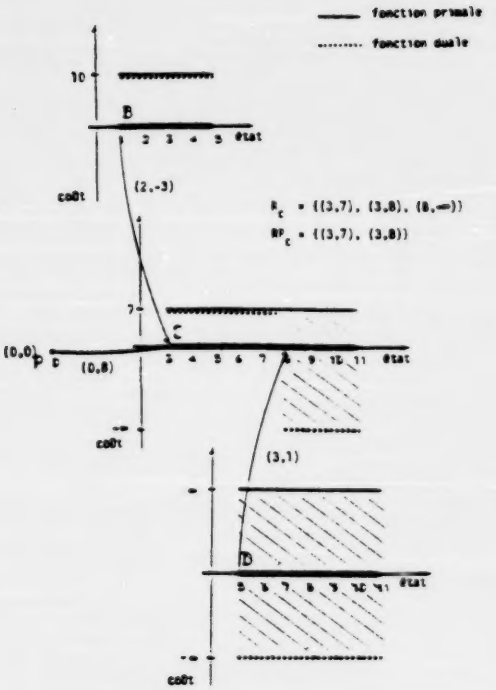


Figure 6.5
Pulling de l'étiquette (3,-∞)
au noeud C

zone d'incertitude comprend tous les états du noeud D. Ce noeud a un seul prédécesseur : la source. Nous obtenons l'étiquette (5,3) et les ensembles $P_D=Q_D=\{(5,3)\}$. L'incertitude est éliminée à ce noeud. Le résultat est illustré par la figure 6.6.

À la quatrième itération, l'ensemble S ne comprend plus que l'étiquette (8,-∞) au noeud C. La zone d'incertitude comprend les quatre états associés aux valeurs 8 à 11. Ce noeud ayant trois prédécesseurs, nous obtenons les étiquettes {(8,8), (8,7), (8,4)}. Les ensembles $P_C=Q_C$ = {(3,7), (8,4)} et l'incertitude est éliminée du noeud et du problème. La solution finale est illustrée à la figure 6.7.

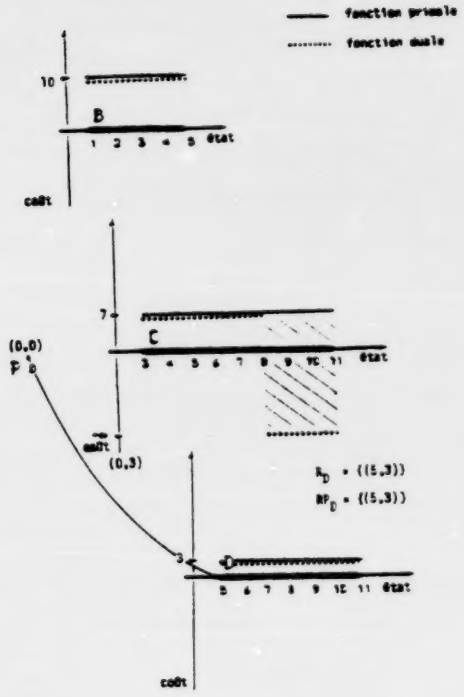


Figure 6.6
Pulling de l'étiquette (5,-∞)
au noeud D

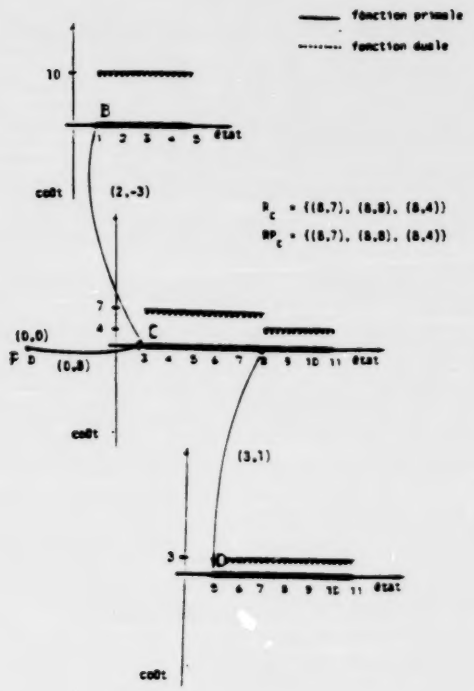


Figure 6.7
Pulling de l'étiquette (8,-∞)
au noeud C

6.7 LE CHOIX DE LA ZONE À EXPLORER

Plus tôt, un exemple de zone à explorer a été donné. Toutefois, aucune définition précise n'a été énoncée. Puisque la zone à explorer débute avec l'apparition d'une différence entre les valeurs des fonctions primale et duale et qu'elle est délimitée par ces deux fonctions, pour définir complètement la zone à explorer, il ne reste qu'à définir l'état de la fin de la zone.

Un choix comme état de la fin de la zone d'incertitude est TR_j = minlex $((b_j^*, \{T_j^m \mid (T_j^m, C_j^m) \in P_j$ et $T_j^m \overset{L}{>} T_j^k\})$ où T_j^k est le premier état de la zone d'incertitude. TR_j correspond à l'état d'arrivée du premier chemin réalisable lexicographiquement postérieur à l'apparition de l'incertitude, TR_j définit la zone à explorer de dimension maximale correspondant à la

zone d'incertitude complète. La zone ainsi définie peut être très grande par rapport à la réduction de la zone d'incertitude assurée par le théorème 6.1. La construction de la borne supérieure est surtout efficace au début de la zone d'incertitude. L'utilisation d'une zone d'exploration beaucoup plus grande que la réduction d'incertitude assurée peut aussi entraîner une série de remplacements avant d'éliminer complètement l'incertitude. Par exemple, si l'étendue de la zone est de n et que la diminution assurée de la zone est de 1. Après la première itération, la dimension de la zone serait de n-1 unités et, après la seconde itréation, la dimension de la zone serait encore de n-2 unités. Dans le pire cas, la dernière partie de l'intervalle peut être traité n fois.

Pour éviter des séries de remplacements, il vaut mieux utiliser une longueur fixe basée sur la réduction assurée de l'incertitude ou une longueur variable basée sur cette valeur. Par exemple :

$$TSUP_1 = \text{minlex}\{T_j^{k+\alpha d_{\cdot j}^{\cdot}}, TR_j\}$$

ou $\alpha > 1$ et $(d_{\cdot j}^{\cdot}, c_{\cdot j}) = \underset{(i,j)\in A}{\text{minlex}} \{(d_{ij}^{\cdot}, c_{ij})\}$.

Ce critère de définition de la zone à explorer garantit qu'aucune partie de la zone d'incertitude ne sera traitée plus de α fois, permettant la mise au point d'un algorithme plus efficace.

Il semblerait raisonnable d'utiliser α=1 pour minimiser la complexité de l'algorithme, toutefois une étude de l'effet de α sur le temps d'exécution effectuée au chapitre précédent pour le cas d'une contrainte supplémentaire a montré que α compris entre 2 et 3 réduisait d'environ 25% les temps d'exécution par rapport à α=1. Cette diminution du temps d'exécution s'explique par une diminution du nombre minimum de remplacement d'étiquettes à effectuer. Toutefois, si α devient trop grand, alors chaque remplacement devient peu efficace. Même si nous ne pouvons prétendre généraliser ces résultats, nous avons choisi d'utiliser α=2 dans notre implantation.

6.8 IMPLANTATION ET COMPLEXITÉ

Nous voulons comparer la complexité en temps et espace mémoire de l'algorithme proposé avec celle d'une généralisation de l'algorithme de marquage permanent présenté au chapitre 4 pour le cas particulier d'une seule contrainte. Les notations utilisées au cours de cette comparaison sont :

L : le nombre de contraintes supplémentaires,

$n = |N|$: le nombre de noeuds dans le graphe G,

$m = |A|$: le nombre d'arcs dans le graphe G,

$v = \max_{i \in N} \{ \prod_{\ell=1}^{L} (b_i^\ell - a_i^\ell + 1) \}$: le nombre maximum d'états à un noeud,

$V = \sum_{i=1}^{N} \prod_{\ell=1}^{L} (b_i^\ell - a_i^\ell + 1)$: le nombre total d'états.

Deux classes d'implantation sont possibles pour les algorithmes de reaching et de pulling. Elles diffèrent par la quantité de mémoire centrale requise pour représenter les états. La première classe demande la conservation en mémoire centrale des étiquettes associées à tous les états possibles dans une table de dimension V permettant un accès direct aux étiquettes. La seconde classe ne conserve en mémoire centrale que les étiquettes efficaces dans un arbre cousu auto-ajustant (SPLAY) mis au point par Sleator et Tarjan [SLEA83] permettant d'accéder à un élément et à son successeur. Sleator et Tarjan montrent que la complexité de toute séquence de t opérations sur un arbre de v étiquettes (avec $t > v$) est d'ordre $t \log v$. Ce résultat permet, en moyenne, d'accéder à une étiquette en temps logarithmique. Selon les résultats obtenus précédemment seulement 5% à 15% des étiquettes sont efficaces et l'utilisation de cet arbre permet une économie appréciable de mémoire.

L'algorithme de reaching pour le cas particulier d'une seule contrainte peut facilement être généralisé et l'analyse de sa complexité faite au chapitre précédent peut facilement être adaptée pour L contraintes. La

complexité pour le cas à une contrainte est d'ordre O(nV) en utilisant une table et d'ordre $O(V^2)$ en utilisant une liste pointée. La seule différence étant que chaque calcul ou comparaison d'étiquettes demande O(L) opérations au lieu de O(1) dans le cas étudié. La complexité de l'algorithme de reaching devient pour la classe d'implantation utilisant une table des états d'ordre O(nVL). Tandis que pour la classe utilisant un arbre auto-ajustant [SLEA 83] ou une liste pointée, la complexité est d'ordre $O(V^2L)$.

La complexité de l'algorithme de pulling dépend de la classe d'implantation utilisée. Toutefois la complexité des quatre premières étapes de l'algorithme est indépendante de la structure de données choisie. Globalement, ces quatre étapes demandent O(m L+nL+VL.logn) opérations.

L'initialisation, lors de l'étape 1, prépare les ensembles P_i et Q_i. La fabrication d'un tas (heap) comprenant les éléments lexicographiquement minimum de chacun des ensembles Q_i demande O(Lnlogn) opérations. Lors de l'étape 1, il est aussi nécessaire de déterminer, par un examen de tous les arcs, l'arc lexicographiquement minimum à chaque noeud, ce qui demande O(m L) opérations. La complexité, en pire cas de l'étape 1, est globalement de O(L.(m +n+nlogn)) opérations.

L'utilisation d'un tas (heap) permettant l'accès rapide à l'élément lexicographiquement minimum de chaque ensemble Q_i permet de simplifier les étapes 2 et 3. Dans le cas simple, le choix de l'étiquette à remplacer se limite à mettre à jour le tas et à accéder à son sommet.

S'il est possible de retarder le remplacement d'une étiquette, les étiquettes dont le remplacement est reporté sont ajoutées (en ordre lexicographique) à une file d'éléments du seau traité.

Dans ce cas, l'étape 2 consiste à définir les limites du seau en trouvant la plus petite étiquette à remplacer en comparant l'étiquette au sommet du tas et la première étiquette de la file. L'étape 3 consiste à mettre à jour le tas et à choisir la prochaine étiquette à remplacer, i.e.

le sommet du tas, s'il appartient au seau traité ou, sinon, un élément de la file.

Dans les deux cas, ces étapes seront exécutées au maximum V fois. Chaque exécution demande O(L.logn) opérations pour la mise à jour du tas et le choix de l'étiquette à traiter. Globalement, les étapes 2 et 3 demandent dans le pire cas O(V.L.logn) opérations.

La quatrième étape peut être exécutée au maximum V fois et demande le calcul de la frontière de la zone à explorer en O(L) opérations. Globalement, l'étape 4 demande en pire cas O(VL) opérations.

La cinquième étape peut être implantée de trois façons, selon la structure de données et l'algorithme utilisés pour tester la dominance.

L'utilisation d'une table de dimension V pour conserver toutes les étiquettes possibles permet un accès direct à chaque étiquette et une vérification facile de la dominance entre deux étiquettes à un noeud. La construction des bornes inférieure et supérieure dans une zone d'incertitude demande le calcul du coût de chacun des chemins en provenance des m prédécesseurs et dont l'état est dans la zone d'incertitude. Ce calcul demande l'accès, en temps constant, aux étiquettes des prédécesseurs. La vérification de la dominance dans une table se limite à vérifier si le coût du chemin est plus petit que le coût actuel. Globalement, chaque étiquette étant traitée au maximum une fois si $\alpha=1$ lors du calcul de l'intervalle d'incertitude, l'étape 5 demande pour le remplacement des étiquettes éliminées des n noeuds O(V.n.L) opérations. Soit la même complexité que l'implantation correspondante de l'algorithme de reaching.

L'utilisation d'un arbre cousu auto-ajustant [SLEA 83] pour conserver les étiquettes efficaces permet, en moyenne, un temps d'accès logarithmique aux étiquettes. L'utilisation d'une table de dimension v pour représenter les étiquettes associées aux états dans la zone d'incertitude permet

de simplifier la vérification de la dominance et de profiter d'une partie des avantages d'une table, tout en conservant l'économie d'espace des arbres. La construction des bornes inférieure et supérieure dans la zone d'incertitude demande le calcul du coût de chacun des chemins en provenance des prédécesseur du noeud et ayant leur étiquette associée dans la zone d'incertitude. L'accès logarithmique aux étiquettes des prédécesseurs demande en moyenne O(logv) opérations. Cette moyenne peut être utilisée puisque le nombre d'accès aux étiquettes est O(v.nombre de successeurs d'un noeud) ce qui est plus grand que O(v). La vérification de la dominance s'effectuant dans la table locale, elle se limite à vérifier si le coût du chemin est plus petit que le coût actuel de l'état associé au chemin. Lors de la fabrication des nouveaux ensembles Q_j et P_j, il y a vérification de la dominance entre les étiquettes de la table et reconstruction de l'arbre cousu auto-ajustant. Globalement, puisqu'il y a au total V étiquettes et puisque chacune de ces étiquettes est traitée au maximum α fois, l'étape 5 demande pour le remplacement des étiquettes Q_j des n noeuds $O(VL\alpha(n\log v+v))$ opérations et $\alpha > 1$.

La complexité globale de la seconde classe d'implantation est d'ordre $O(VL.(n\log v+v))$ pour le pulling et d'ordre $O(V^2L)$ pour le reaching. Puisque $O(n\log v+v) < O(V)$, la complexité de cette implantation du pulling est moindre que celle de l'implantation pour le reaching.

La troisième implantation possible utilise aussi un arbre cousu auto-ajustant pour conserver les étiquettes efficaces. Toutefois, le test de dominance n'est pas effectué dans une table mais plutôt en utilisant un algorithme de calcul du minimum d'un ensemble de vecteurs (i.e. d'étiquettes multidimensionnelles) [KUNG75]. La complexité de cet algorithme est, en pire cas, d'ordre $O(\alpha v(\log v)^{L-2})$ à chaque noeud. Globalement, pour les n noeuds, le calcul de la dominance demande $O(V(\log v)^{L-2})$ opérations et l'accès aux prédécesseurs $O(VL\alpha\log v)$ opérations. L'étape 5 demande, pour cette implantation, $O(V\alpha(nL\log v+(\log v)^{L-2}))$ opérations ($\alpha>1$).

Dans le cas où L<3, Kung, Luccio et Preparata [KUNG 75] présentent des algorithmes pour le calcul du minimum d'un ensemble de vecteurs de dimension 2 et 3, i.e. L=1,2. Ces algorithmes fonctionnent en temps O(vlogv). La complexité de l'étape 5 est alors d'ordre $O(\alpha V \log v(nL+v))$ pour le cas L=1 ou 2.

La complexité globale de la troisième classe d'implantation est d'ordre $O(V(nL\log v+(\log v)^{L-2}))$ pour le pulling et d'ordre $O(V^2L)$ pour le reaching (n.b. la comparaison est effectuée avec la deuxième implantation du reaching, la troisième implantation du reaching n'offrant aucun avantage). Encore une fois, la complexité de l'implantation du pulling est moindre que celle du reaching.

Tableau 6.1

Sommaire de la complexité des algorithmes de reaching et de pulling multidimensionnel.

implantation	espace-mémoire		temps
reaching avec table	$\Omega(m+V)$		$O(nD)$
reaching avec liste pointée (ou splay)	$O(m+V)$		$O(V^2L)$
pulling avec table	$\Omega(m+V)$		$O(nV)$
pulling avec splay et petite table	$O(m+V)$		$O(VL(n\log v+v))$
pulling avec splay et algorithme de dominance	$O(m+V)$	(L>3)	$O(V(nL\log v+(\log v)^{L-2}))$
		(L=1,2)	$O(V\log v(nL+v))$

6.9 RÉSULTATS NUMÉRIQUES

L'algorithme de pulling présenté à la section 6.5 a été implanté en FORTRAN77 selon deux des trois versions de la section 6.8 en utilisant $\alpha=2$. PULL1 utilise des arbres cousus auto-ajustants pour conserver les étiquettes efficaces et utilise une table pour vérifier la relation de dominance entre les étiquettes. PULL2 utilise aussi des arbres cousus auto-ajustants pour conserver les étiquettes efficaces, mais utilise un algorithme spécialisé pour vérifier la dominance. Les deux implantations partagent les mêmes sous-routines, sauf celles vérifiant la dominance. Les deux versions ont été compilées sur le compilateur FTN5 avec l'option OPT=2 sur les ordinateurs CDC CYBER 173 de l'Université de Montréal. La version PULL1 ne peut être utilisée que si le nombre d'états à chaque noeud, i.e. la taille de la table utilisée pour vérifier la dominance, n'est pas trop grand. La comparaison entre les deux versions ne pourra donc être effectuée que sur de tels problèmes. Il n'existe pas d'implantation de l'algorithme de reaching pour le PPCCCS. Toutefois, la comparaison effectuée au chapitre V entre le pulling et le reaching pour le cas d'une seule contrainte supplémentaire montrait que le pulling était légèrement plus rapide que le reaching.

La comparaison entre les deux implantations se fera sur des problèmes de type fabrication de journées de travail de conducteurs d'autobus urbains. Nous voulons produire la journée de travail débutant entre les temps A et B, et respectant trois ou cinq contraintes supplémentaires. Une journée de travail comporte de une à plusieurs tâches séparées par des pauses et doit respecter les contraintes suivantes :

1. le nombre de tâches est limité;
2. la durée totale de la journée (i.e. durée combinée des tâches exécutées et les pauses) est limitée;
3. le nombre d'heures de travail (i.e. durée totale des tâches exécutées) est limité.

De plus, pour les problèmes avec cinq contraintes supplémentaires, deux autres ressources sont contraintes.

Pour chacune de ces contraintes, la quantité de la ressource déjà utilisée pour atteindre une tâche doit appartenir à la fenêtre de réalisabilité associée à la tâche et à cette ressource. Dans notre type de problèmes, dans deux cas (nombre de tâches exécutées, nombre d'heures travaillées) la fenêtre est de la forme [0..bsup]. Dans les trois autres cas (durée totale et les ressources 4 et 5), la fenêtre est adaptée à chaque tâche. La fenêtre sur la durée totale pour une tâche débutant au moment d est [d-b..d-a].

L'ordre des éléments à l'intérieur des vecteurs $(d_{ij}^1,\ldots,d_{ij}^L)$ est important pour déterminer l'ordre lexicographique et cet ordre détermine l'efficacité de l'algorithme. Nous avons donc choisi, comme premier élément du vecteur, l'élément ℓ tel que $\Theta^\ell = \left(\frac{b_j^\ell - a_j^\ell + 1}{\min\limits_{(i,j)\in A} d_{ij}^\ell} \right)$ est minimum. De façon à minimiser le nombre maximum de pulling à chaque noeud, Θ^ℓ représente le nombre maximum de seaux généralisés si d_{ij}^ℓ est le premier élément du vecteur.

Nous avons produit deux séries de cinq problèmes pour les quatre tailles de problèmes N = 100, 250, 500, 1000. Les caractéristiques de ces problèmes $[a_i^\ell, b_i^\ell]$, $i\in N$, $\ell=1,\ldots,5$ et c_{ij}, d_{ij}^ℓ, $(i,j)\in A$, $\ell=1,\ldots,5$ ont été distribuées selon des lois uniformes de façon à simuler des problèmes de fabrication de journées. La première série représente des problèmes où les données ont été arrondies pour diminuer la quantité de mémoire v requise à chaque noeud, tandis que dans la seconde série, les données sont plus précises. Ceci permet aussi d'étudier l'effet de v sur le comportement des deux implantations. Nous avons doublé les séries en extrayant aussi des problèmes avec L=3.

Les tableaux 6.2 et 6.3 contiennent les descriptions des problèmes générés. $|N|$ est le nombre de noeuds dans chaque problème. $|A|$ est le nombre moyen d'arcs réalisables dans le réseau. Il est à noter que le nombre d'arcs réalisables est plus grand dans les problèmes avec trois contraintes que dans ceux avec cinq contraintes. Ceci est dû aux vérifications de réalisabilité (6.6) Supplémentaires. v est le nombre d'états à un noeud alors que V est le nombre total d'états. V est le nombre maximum d'étiquettes possibles. $|E_S|$ est le nombre d'étiquettes associées à des chemins efficaces dans la solution finale. Il est fréquent que le nombre d'étiquettes présentées dans la solution lors de l'exécution de l'algorithme soit supérieur à $|E_S|$.

Les statistiques d'exécution présentées dans les tableaux 6.2 et 6.3 sont le nombre de pulling et les temps d'exécution. L'ordinateur utilisé ne permettait pas de conserver en mémoire centrale la totalité des noeuds, des arcs et des étiquettes. Les noeuds et les étiquettes sont conservés en mémoire centrale, alors que les arcs sont divisés en pages et que l'on ne conserve qu'une seule page en mémoire centrale. Lors d'un défaut de page, i.e. lorsque l'exécution de l'algorithme nécessite des arcs d'une autre page, cette nouvelle page est amenée en mémoire centrale. Ce processus peut se comparer à celui de la mémoire virtuelle sur d'autres ordinateurs. Le temps de gestion de la mémoire (temps ES) est comptabilisé séparément du temps d'exécution de l'algorithme (temps UC). Toutefois, cette gestion étant effectuée par une partie du programme commune aux deux implantations, aucune différence significative n'a été observée entre les temps de gestion de la mémoire des deux implantations.

Les statistiques d'exécution des 16 séries de problèmes sont contenues dans les tableaux 6.2 et 6.3. La première constatation est que PULL2 a pu résoudre 15 des 16 séries de problèmes, alors que PULL1 n'a pu résoudre, faute de mémoire centrale suffisante, que 11 des 16 séries. La série

TABLEAU 6.2

Description des problèmes					PULL1			PULL2		
|N|	|A|	V	$\|E_s\|$	#PULL	Temps ES (sec.)	Temps UC (sec.)		Temps ES (sec.)	Temps UC (sec.)	
					Moy.	Max	Moy	Moy.	Max	Moy
100	1566	9224	164	101	0,000	1,551	1,214	0,000	1,011	0,890
250	9096	23086	526	251	0,023	6,029	4,612	0,024	5,014	4,294
500	36241	46210	1290	501	0,348	21,717	16,055	0,349	23,071	17,411
1000	144082	92336	2998	1001	1,830	86,674	59,645	1,846	111,348	69,747

Statistiques d'exécution des deux implantations pour des problèmes de trois contraintes et avec v variant entre 1 et 220 (moy = 92).

|N|	|A|	V	$\|E_s\|$	#PULL	Temps ES (sec.)	Temps UC (sec.)		Temps ES (sec.)	Temps UC (sec.)	
					Moy.	Max	Moy	Moy.	Max	Moy
100	1746	42724	348	101	0,000	4,953	3,235	0,000	1,253	1,097
250	10282	107166	1657	251	0,070	17,890	10,509	0,071	13,732	6,973
500	40836	214416	4578	501	0,532	60,657	34,586	0,438	77,735	31,816
1000	espace mémoire insuffisant									

Statistiques d'exécution des deux implantations pour des problèmes de trois contraintes et v variant entre 1 et 1000 (moy = 427)

TABLEAU 6.3

Description des problèmes					PULL1			PULL2		
|N|	|A|	V	$\|E_s\|$	#PULL	Temps ES (sec.)	Temps UC (sec.)		Temps ES (sec.)	Temps UC (sec.)	
					Moy.	Max	Moy	Moy.	Max	Moy
100	582	133833	83	101	0,000	18,756	10,522	0,000	0,602	0,456
250	3364	325658	401	251	0,000	47,211	27,211	0,000	2,479	2,062
500	12030	646179	1154	501	0,095	97,826	55,956	0,096	9,677	7,050
1000	46889	1295252	2913	1001	0,510	205,787	121,025	0,543	39,681	26,371

Statistiques d'exécution des deux implantations pour des problèmes de cinq contraintes et avec v variant entre 1 et 9200 (moy = 1298)

|N|	|A|	V	$\|E_s\|$	#PULL	Temps ES (sec.)	Temps UC (sec.)		Temps ES (sec.)	Temps UC (sec.)	
					Moy	Max	Moy	Max	Max	Moy
100	641	602812	123	101	espace mémoire insuffisant			0,000	0,686	0,517
250	3742	1511336	729	251				0,000	3,744	2,592
500	13423	2998202	2513	501				0,121	22,712	10,959
1000	52978	6009189	7641	1001				0,636	104,709	46,806

Statistiques d'exécution des deux implantations pour des problèmes de cinq contraintes et v variant entre 1 et 46000 (moy = 6012)

qu'aucune des implantations n'a pu résoudre demandait beaucoup plus de mémoire centrale que les autres. Les deux implantations semblent être complémentaires. PULL1 est légèrement meilleur que PULL2 sur les problèmes avec L=3 et ou v est petit, cet avantage disparaît lorsque v grandit. Dans le cas ou L=5, la domination est claire, PULL2 demande de 5% à 20% du temps de calcul de PULL1 et moins de mémoire centrale.

Il est possible d'expliquer la différence de temps d'exécution de PULL1 et PULL2 par la nature des variables présente dans l'analyse en pire cas de leur complexité. Pour PULL1, cette complexité est d'ordre $O(VL(n\log v+v))$, et d'ordre $O(V(nL\log v+(\log v)^{L-2}))$ pour PULL2. La partie $O(VLn\log v)$ est identique pour les deux implantations, seule la partie $O(VLv)$ de PULL1 diffère de la partie $O(V(\log v)^{L-2})$ de PULL2. La vérification de la dominance dans PULL1 demande au moins $O(v)$ opérations alors que dans le cas de PULL2, elle en demande $O(K(\log k)^{L-2})$ où k est le nombre de chemins examinés ($k<v$). Dans le cas où v est grand et k est petit, $O(v)$ est souvent plus grand que $O(K(\log k)^{L-2})$, ce qui explique que PULL2 soit plus rapide que PULL1 dans le cas ou L=5.

Le comportement de l'algorithme PULL2 est surprenant à un autre point de vue : son temps d'exécution diminue lorsque l'on augmente le nombre de contraintes de 3 à 5 et que le nombre d'états V a considérablement augmenté. Cette diminution s'explique par la réduction du nombre d'arcs réalisables dans les problèmes entraînant une réduction du nombre d'états examinés. Le nombre d'opérations de PULL2 dépend du nombre d'étiquettes examinées plutôt que du nombre d'états V. PULL2 permet la résolution de problèmes fortement contraints en un temps raisonnable.

6.10 CONCLUSION

Le nouvel algorithme pour les problèmes de plus court chemin avec contraintes supplémentaires a été développé pour résoudre les sous-problèmes de routing et d'ordonnancement. Il permet aussi la résolution de

problèmes de type sac alpin à une à plusieurs ressources en utilisant des fenêtres $[0,b^{\ell}]$ à tous les noeuds. De plus, nous avons déjà vu que cet algorithme peut être utilisé pour réoptimiser après des augmentations de coûts.

Le pulling qui trouve la solution à un noeud en consultant la solution de plusieurs noeuds est mieux adapté à une implantation sur un ordinateur effectuant des opérations en parallèle que le reaching qui modifie la solution de plusieurs noeuds lors de la consultation de la solution d'un noeud.

L'examen des résultats numériques montre de plus une conclusion surprenante : pour PULL2, le temps de calcul semble plus influencé par le nombre de solutions au problème que par le nombre d'états dans le problème. En effet, l'ajout de deux autres contraintes aux problèmes avec L=3 augmente beaucoup le nombre d'états V mais diminue le nombre d'arcs et de solutions du problème. On observe alors une diminution du temps d'exécution entre les problèmes avec L=3 et L=5 quand le nombre de contraintes augmente. Ceci et une possible implantation sur ordinateur multi-processeur ouvre la voie à la résolution de problèmes complexes et très contraints de grande taille en temps raisonnable.

CHAPITRE VII

LA FABRICATION D'HORAIRES DE TRAVAIL PAR GÉNÉRATION DE COLONNES : IMPLANTATION ET RÉSULTATS NUMÉRIQUES

Au chapitre III, la méthode de génération de colonnes pour la fabrication d'horaires de travail a été décrite. Les chapitres subséquents décrivaient une méthode de résolution pour le problème de plus court chemin avec contraintes supplémentaires. L'union de ces méthodes permet la fabrication d'horaires de travail. Dans ce chapitre, nous décrirons comment a été réalisée l'implantation permettant la fabrication d'horaires de travail pour des problèmes réels provenant de deux villes : une ville américaine et une ville anglaise. Toutes les implantations ont été réalisée en FORTRAN 77 sur les CYBER 173 du centre de calcul de l'Université de Montréal.

La première partie du chapitre traitera principalement de la formulation des sous-problèmes de plus court chemin avec contraintes supplémentaires utilisés pour fabriquer les journées de travail. La seconde partie décrira la résolution de problèmes réels. Pour chaque ville nous décrirons la convention collective en vigueur, le problème à résoudre et les résultats obtenus. Ces derniers seront comparés aux résultats obtenus par l'utilisation du logiciel HASTUS.

7.1 LES SOUS-PROBLÈMES

Au chapitre III, nous avons modélisé la fabrication d'une journée de travail comme la résolution d'un sous-problème de plus court chemin avec contraintes supplémentaires. Toutefois, même pour un problème de taille normale (200-300 tâches), la résolution d'un tel sous-problème peut être très difficile. Il est préférable de scinder le sous-problème original en plusieurs sous-problèmes de taille moindre. La division du sous-problème original s'effectuera selon deux critères :

- par division des journées de travail en types (i.e. journées continues vs journées brisées);
- par division des journées de travail par heure de début (journées débutant avant 8h00, entre 8h01 et 12h00, etc.).

Cette division n'est pas nouvelle, elle correspond à la définition des différents types de journées (continue, brisée, continue-tôt, continue-tard, etc.) dans certains logiciels. Cette division des sous-problèmes est efficace car elle évite l'étude de plusieurs solutions non-réalisables. Par exemple, soit une tâche i de 13h00 à 14h00 et une tâche j de 16h00 à 17h00, ces deux tâches ne peuvent faire partie d'une même journée de travail que si la tâche i appartient à la première pièce de travail et la pièce j à la seconde pièce de travail. Dans le sous-problème original, la tâche i peut faire partie de la première ou de la seconde pièce d'une journée et nous étudierons dans les deux cas la possibilité d'exécuter aussi la pièce j dans la même journée. Si les sous-problèmes sont divisés, pour le sous-problème fabriquant les journées débutant très tôt (i.e. les journées ou la tâche i fait partie de la seconde pièce), nous avons a priori éliminé la possibilité d'inclure dans ces journées la tâche j. Ceci permet d'éviter d'étudier une journée non-réalisable. L'arc entre la fin de la tâche i et le début de la tâche j n'étant présent que dans les sous-problèmes où i peut faire partie de la première pièce de travail.

Les sous-problèmes sont caractérisés par un type de journée de travail (continue ou brisée) et par un intervalle de début possible des journées. La division peut être effectuée par l'usager qui décrit chaque sous-problème (continue-tôt, continue-tard, continue-médiane, brisée-tôt, brisée-tard, etc.) de façon semblable à la division proposée par Ward et al. [WARD 81]. La division des sous-problèmes peut aussi être effectuée automatiquement. La longueur maximale d'un intervalle, PERIODE, est comprise entre la longueur minimale et la longueur maximale d'une pièce de travail. Dans un première étape, les tâches sont partitionnées en trois groupes :

1) les tâches débutant avant la période de pointe du matin,

2) les tâches débutant entre les deux périodes de pointe,

3) les tâches débutant après la période de pointe du soir.

Dans une seconde étape, chaque groupe est divisé en sous-groupes tel que dans chaque sous-groupe la différence entre le premier et le dernier début de tâches soit inférieure à PERIODE. Chaque sous-groupe correspond un sous-problème ou tous les éléments peuvent débuter une journée de travail.

7.2 SOLUTION INITIALE ET SOUS-PROBLÈMES INITIAUX

Si on ne dispose pas d'une solution initiale composée de journées de travail réalisables, il faut utiliser une solution initiale composée de pièces vacantes. Cette solution initiale possède un désavantage majeur : les valeurs des variables duales sont peu significatives et presque toutes les journées réalisables ont un coût marginal négatif. La résolution des premiers sous-problèmes est très lente car beaucoup de journées de travail sont produites et le temps de résolution des sous-problèmes dépend beaucoup du nombre de journées de travail produites. À cette étape, il est facile de réduire le nombre de journées produites (et le temps de résolution) en forçant temporairement des tâches consécutives à faire partie de la même pièce de travail. Ceci permet de réduire considérablement le nombre de journées de travail à étudier et permet d'obtenir rapidement une solution initiale.

7.3 UN PROBLÈME PROVENANT D'UNE VILLE AMÉRICAINE

Ce premier problème à résoudre provient d'une ville américaine. Dans cette ville, les horaires de travail sont fabriqués séparément pour chaque ligne. Le problème à résoudre est composé de 25 blocs de véhicules totalisant 347 heures de travail. La figure 7.1 illustre l'histogramme du nombre d'autobus en service à chaque moment de la journées sur la ligne.

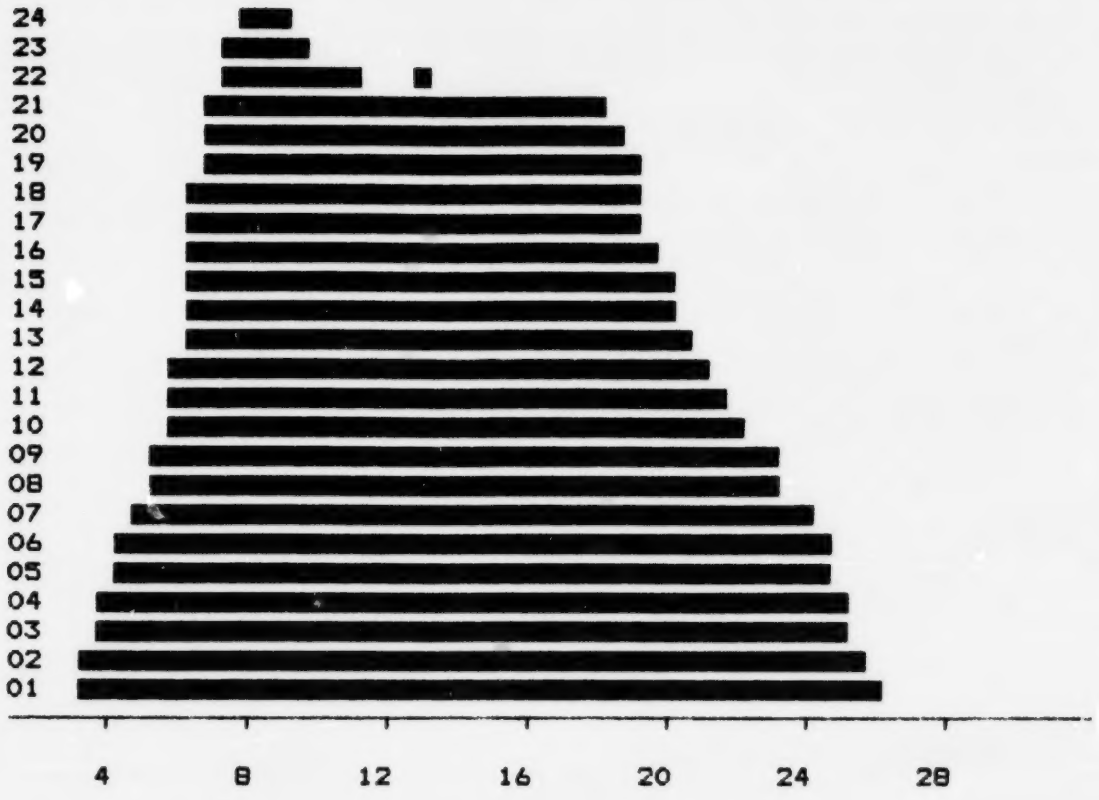


Figure 7.1 : Courbe de service du premier problème

La convention collective conclue par la commission de transport locale et les conducteurs d'autobus divise les journées en deux types : les journées "continues" qui sont, en fait, des journées de deux pièces ayant une petite pause (d'une durée comprise entre 30 et 59 minutes) et des journées brisées qui sont des journées de trois pièces ou des journées de deux pièces dont la durée de la pause excède 59 minutes. Toutefois, les journées de trois pièces sont interdites sur notre ligne. Une des particularités de cette convention est d'inclure dans la durée d'une pièce de travail les primes de début ou de fin de journées, le temps de déplacement depuis le garage en plus du temps de travail sur la ligne. L'autre particularité de cette convention collective est d'exiger le retour au garage si le point de relève associé au début d'une pause est différent de celui associé à sa fin.

Les autres caractéristiques de la convention collective peuvent être résumées par les paramètres suivants :

- longueur maximale d'une pièce de travail : 6 heures
- amplitude maximale d'une journée : 13 heures
- amplitude maximale sans pénalité : 11 heures
- pénalité pour grande amplitude : 0,5
- maximum du temps payé pour une journée-continue : 11 heures
- maximum du temps payé pour une journée-brisée : 13 heures
- prime de début de journée : 10 minutes
- prime de fin de journée : 5 minutes
- temps payé garanti pour une journée de travail : 8 heures
- pénalité pour temps supplémentaire : 0,5

De plus, il existe deux contraintes additionnelles. La première exige qu'au moins 50% des journées de travail soient continues. La seconde exige que 65% des journées de travail aient une amplitude de 10 heures et

moins. La convention collective permet la construction de journées vacantes formées de tâches consécutives d'un bloc aux conditions suivantes :

- durée maximale d'une pièce vacante : 6 heures
- pénalité pour pièce vacante : 0,5
- temps payé garanti pour une pièce vacante : 2 hrs 40 min.

Ce problème a été soumis par l'exploitant pour permettre d'évaluer les performances relatives d'HASTUS et de sa méthode actuelle (dénotée ACTUELLE). Cette méthode est basée sur la résolution d'un problème de recouvrement tel qu'exposé au chapitre II. Pour effectuer la comparaison entre les deux méthodes, il fut décidé que les répartiteurs de l'exploitant résoudraient le problème en utilisant la méthode actuelle et que l'équipe de HASTUS en ferait autant en utilisant HASTUS. La comparaison porterait sur les caractéristiques de la solution (coût, nombre de journées, etc.).

Le problème original comporte 355 tâches. La distribution de la longueur des tâches est très simple : 183 tâches de moins de 20 minutes, 171 tâches de plus de 90 minutes et plus et une tâche de 24 minutes. De plus, la distribution des tâches dans les blocs est alternée : une tâche courte, une tâche longue et une tâche courte. Pour réduire la taille du problème, nous avons décidé de fusionner chaque tâche courte à une des deux tâches longues l'entourant. On choisit aléatoirement à quelle tâche longue on joint la tâche courte. Les tâches définitives peuvent être formées d'une tâche longue, d'une tâche longue et d'une courte ou d'une longue et de deux courtes. Ceci permet l'obtention de tâches de longueurs variées. Après avoir fusionné les tâches courtes et éliminé les points de relève illégaux, nous obtenons un problème de 167 tâches.

La résolution du problème s'est effectuée en deux phases. La première phase consistant à trouver une solution réalisable en utilisant la formulation originale des contraintes additionnelles (3.12-3.16). La seconde

utilisant la formulation modifiée de ces contraintes pour obtenir des contraintes actives lors du branchement.

Nous avions posé un temps maximum pour la première phase. Nous avions obtenu une solution au problème perturbé d'un coût de 4 066,34 $. Cette solution a été obtenue en 1350 secondes de temps CPU dont 750 pour la génération de colonnes. Le programme a résolu 100 sous-problèmes de plus court chemin avec contraintes supplémentaires produisant 1 712 journées de travail réalisables.

La seconde passe a permis dans un premier temps l'obtention de la solution continue au problème original au coût de 4 033,90 $. Une solution entière est ensuite obtenue en fixant une paire de tâches selon la méthode de branchement décrite à la section 4 du chapitre III. Nous avons choisi d'accélérer l'obtention d'une solution entière en fixant plusieurs paires de tâches simultanément. Au maximum cinq paires peuvent être fixées simultanément.

Tableau 7.1

Caractéristiques des noeuds étudiés par la méthode de branchement pour le premier problème.

Niveau	Nombre de variables fractionnaires	Valeur objectif	Nombre de paires éligibles
0	102	4033,90	26
5	86	4033,90	20
10	89	4034,29	14
15	37	4035,37	2
17	0	4036,33	0

La tableau 7.1 comprend les caractéristiques des noeuds étudiés par la méthode de branchement lors de l'obtention d'une solution entière. Le niveau indique le nombre de paires de tâches fixées. Dans la seconde colonne, le nombre de variables fractionnaires est indiqué. La troisième colonne contient la valeur de la fonction objective du problème résultant. La quatrième colonne contient le nombre de paires de tâches consécutives pouvant être fixées ensemble. Pour pouvoir être fixées ensemble des tâches consécutives doivent, dans la solution actuelle, respecter :

$$0.5 < \sum_{j \in T(j_i, j_{i+1})} x_j < 1.0$$

où $T(j_i, j_k)$ est l'ensemble des journées de travail recouvrant consécutivement les tâches i et k. Étant donné la faible différence entre la valeur de la fonction objective de l'optimum continu et celle de la solution entière, nous n'avons pas jugé nécessaire d'explorer complètement l'arbre des branchements.

La méthode de branchement a permis l'obtention rapide d'une solution entière. Ce succès est dû au faible nombre de tâches par journée de travail, 3,7 en moyenne qui interdit la présence de nombreuses paires éligibles.

L'obtention d'une solution entière à partir de la solution initiale calculée lors de la première passe a demandée au total 2 652 secondes CPU. Le programme a résolu 207 sous-problèmes de plus court chemin avec contraintes supplémentaires produisant 1 214 colonnes en 1 595 secondes CPU. De plus, la résolution du problème de recouvrement par l'algorithme du simplexe a demandé 676 secondes CPU.

Tableau 7.2

Comparaison des caractéristiques des solutions pour le premier problème.

MÉTHODE	HASTUS	ACTUELLE	GÉNÉRATION DE COLONNES
Coût ($)	4 073,50 $	4 069,50 $	4 036,33 $
Nombre de journées			
- totales	45	45	45
- continues	32	36	23
- brisées	13	9	22
Nombre de pièces vacantes	0	0	0
Répartition du temps payé			
Temps utile	347h03	347h03	347h03
Primes	11h15	11h15	11h15
Déplacement	4h10	4h10	4h15
Pauses payées	31h06	31h54	29h27
Garanties	5h24	4h30	4h39
Temps supplémentaire	7h49	7h55	7h06,5
Amplitude	0h34	0h10	0h02,5
Total	407h21	406h57	403h38
Efficacité (utile/payé)	85,2 %	85,3 %	86,0 %

Nous ne disposons pas des solutions obtenues lors de la résolution par les deux autres méthodes. Toutefois, nous disposons de statistiques sur ces solutions. Le tableau 7.2 contient les caractéristiques des trois solutions. La génération de colonnes a permis d'épargner 33,17 $ (0,8 %) par rapport au coût de la solution actuelle et 37,17 $ (0.9 %) par rapport à celui de la solution de la méthode HASTUS. La principale raison de cette économie est le nombre plus élevé de journées brisées dans la troisième solution. Ceci entraîne une diminution des pauses payées car les pauses des

journées brisées sont payées à demi. Cette différence explique une bonne partie de la différence de coût. L'autre partie s'explique par une diminution du temps supplémentaire payé. Même si la méthode actuelle et la méthode de génération de colonnes utilisant tous deux un problème de recouvrement pour trouver une solution. La génération de colonnes a permis une amélioration de 0,7% de l'efficacité moyenne des journées de travail.

La magnitude des sommes épargnées peut sembler faible mais il ne faut pas oublier qu'il s'agit de l'épargne pour une seule journée sur une seule ligne. Le même horaire sera utilisé durant longtemps (ex. 6 mois) multipliant l'épargne par le nombre d'utilisations.

7.4 UN PROBLÈME PROVENANT D'UNE VILLE ANGLAISE

Le second problème à résoudre provient d'une ville anglaise. La modélisation de ce problème est quelque peu différente de celle du problème précédent. Cette convention collective introduit la notion de demi-journée. Une demi-journée est composée d'une pièce ou de deux pièces jointes par une courte pause. De plus, la durée d'une demi-journée est limitée à 4 heures et 30 minutes. Ceci modifie un peu la formulation du problème de plus court chemin avec contraintes supplémentaires. Une des ressources est modifiée. Les trois premières ressources (nombre de pièces de travail, amplitude et temps payé) demeurent inchangées mais la dernière (temps travaillé) devient le temps écoulé depuis le début de la présente demi-journée. Cette modification permet de vérifier la durée de chaque demi-journée.

Le problème à résoudre est composé de 20 blocs de véhicules totalisant 310 heures de travail. Ce problème comprend deux lignes d'autobus. La figure 7.2 illustre l'histogramme du nombre d'autobus en service à chaque moment de la journée.

Il n'y a qu'un type de journée selon la convention collective. Ce sont des journées ayant au moins une pause-repas pour permettre au

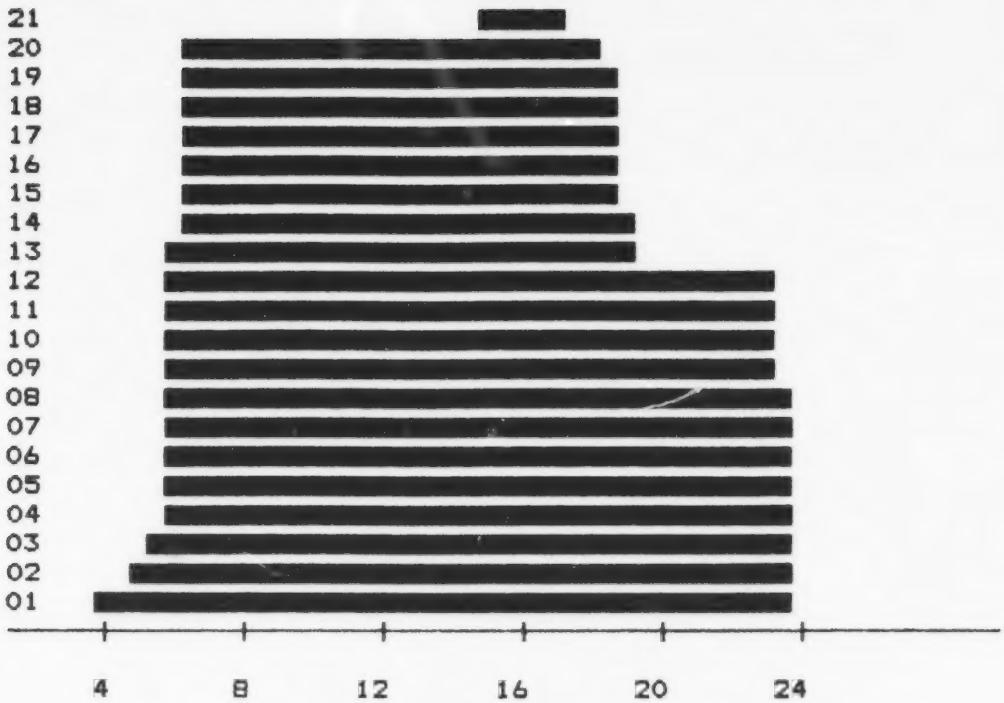


Figure 7.2 : Courbe de service pour le second problème.

conducteur de retourner à une des cantines. De plus, une des demi-journées peut être composée de deux pièces de travail. La seule particularité de la convention collective est de payer les conducteurs selon l'amplitude de la journée, i.e. toutes les parties de la journées (primes, pauses et pièces de travail) sont payées au même taux horaire. Toutes les caractéristiques de la convention collective sont résumées par les paramètres suivants :

- longueur minimale d'une pièce de travail : 1 heure
- longueur maximale d'une pièce de travail : 4 hrs 30 min.
- amplitude maximale d'une journée : 8 hrs 30 min.
- durée minimale d'une pause-repas (ampli. ≤ 8h00) : 35 minutes
  (déplacement exclus) (ampli. > 8h00) : 45 minutes
- dernière heure de début d'une pause-repas : 20 hrs 15 min.

De plus, il faut vérifier que la cantine est ouverte au début de la pause-repas. Une des cantine ouvre ses portes à 6h00 et l'autre à 8h30.

- prime de début de journée (au garage) : 25 minutes
  (ailleurs) : 15 minutes
- prime de fin de journée (au garage) : 25 minutes
  (ailleurs) : 15 minutes
- durée minimale d'une petite pause : 5 minutes
  (déplacement inclus)

L'exploitant impose aussi une contrainte additionnelle sur le nombre de journées de travail faisant partie de l'horaire.

Ce problème a été soumis par l'exploitant pour permettre d'évaluer les performances relatives d'HASTUS et de sa méthode actuelle (dénotée ACTUELLE). La comparaison s'effectuait dans des conditions similaires au test sur le problème précédent. Les critères de comparaison étaient similaires. L'exploitant demandait toutefois que la solution de HASTUS requière

au moins un conducteur de moins que sa solution obtenue par la méthode actuelle.

Le problème original comporte 498 tâches. La durée moyenne d'une tâche est de 37 minutes. Nous avons décidé de réduire la taille du problème en joignant des tâches de façon à obtenir de plus grosses tâches. Notre but est d'obtenir des tâches d'une durée comprise entre 60 et 135 minutes. La limite supérieure a été choisie de façon à ce qu'il soit possible d'utiliser deux tâches consécutives pour former une pièce de travail. Après cette réduction de taille et l'élimination des points de relève illégaux, nous obtenons un problème de 235 tâches.

Nous disposons de la solution obtenue par la méthode HASTUS pour initialiser notre méthode. L'obtention de la solution optimale du problème de recouvrement avec contraintes d'intégrité relaxées a demandé un temps assez considérable : 8 968 secondes de temps CPU. Le programme a résolu 418 sous-problèmes de plus court chemin avec contraintes supplémentaires produisant 12 495 journées de travail en 3 887 secondes de CPU. L'algorithme du simplexe a demandé 3 819 secondes de CPU. Nous reviendrons plus loin sur les raisons probables de ces temps de calcul.

La solution HASTUS utilisée comme solution initiale avait un coût de 4 017,67 $. Notre solution continue a un coût de 3 901,62 $. Il s'agit d'une bonne amélioration. Lors de la seconde phase, l'obtention d'une solution entière, nous avons utilisé la même méthode d'accélération du branchement en fixant simultanément plusieurs paires de tâches simultanément. Au maximum dix paires de tâches peuvent être fixées simultanément.

Le tableau 7.3 comprend les caractéristiques des noeuds étudiés par la méthode de branchement en vue de l'obtention d'une solution entière. Le nombre de paires fixées est beaucoup plus grand que dans le cas du premier problème. Le nombre de paires éligibles est plus grand et puisqu'une

journée de travail peut compter jusqu'à six tâches (4,5 en moyenne), il existe beaucoup de paires possibles. Ceci rend l'obtention d'une solution entière plus difficile.

Tableau 7.3

Caractéristiques des noeuds étudiés par la méthode de branchement pour le second problème.

Niveau	Nombre de variables fractionnaires	Valeur objectif	Nombre de paires éligibles
0	175	3901,62	69
10	155	3904,92	57
20	153	3908,65	54
30	120	3911,32	37
40	93	3912,99	24
50	81	3918,60	20
60	82	3923,68	17
70	39	3928,69	4
73	14	3929,38	2
75	0	3930,14	0

L'obtention d'une solution entière à partir de l'optimum relaxé a demandé 10 426 secondes de temps CPU. Le programme a résolu 653 sous-problèmes de plus court chemin avec contraintes supplémentaires produisant 5 778 journées de travail en 6 423 secondes. La résolution du problème de recouvrement par l'algorithme de simplexe a demandé 3 737 secondes de temps CPU.

Même s'il existe un écart de 22,48 $ entre la solution entière et la solution continue, la taille de l'arbre de branchement nous a fait renoncer à l'exploration de l'arbre.

Tableau 7.4

Comparaison entre les caractéristiques des solutions pour le second problème.

MÉTHODE	ACTUELLE	HASTUS	GÉNÉRATION DE COLONNES
Coût ($)	4 025,33 $	4 017,67 $	3 930,14 $
Nombre de journées			
- totales	53	51	51
- 2 pièces	52	50	47
- 3 pièces	1	1	4
Nombre de pièces vacantes	0	2	2
Répartition du temps payé			
Temps utile	309h45	309h45	309h45
Repas	50h22	44h34	34h51
Primes	32h00	32h00	32h20
Déplacements*	10h25	11h38	13h19
Temps supplémentaire	0h00	3h49	2h46
Total	402h32	401h46	393h01
Efficacité (utile/payé)	77,0 %	77,1 %	78,9 %

* y compris les petites pauses des journées de 3 pièces.

Pour ce problème nous disposons à la fois des solutions obtenues lors de la résolution par les deux autres méthodes et des caractéristiques des solutions. Le tableau 7.4 contient les caractéristiques des trois solutions. La génération de colonnes a permis d'épargner 95,19 $ (2,4%) par rapport au coût de la solution actuelle et 87,53 $ (2,2%) par rapport au coût de la solution HASTUS. La solution de la méthode de génération de colonnes respecte le désir de l'exploitant de diminuer le nombre de conducteurs requis. La solution de la méthode de génération de colonnes contient 3 journées de trois pièces de plus que les deux autres solutions.

Toutefois, la différence majeure entre les trois solutions n'est pas à ce niveau. Il faut examiner la distribution de la durée des repas (tableau 7.5) et de l'amplitude (tableau 7.6) pour mieux comprendre les différences entre les solutions.

Avant de comparer les solutions, nous voulons souligner qu'il existe une discontinuité dans le coût des journées de travail : la durée minimale du repas pour les journées de 8 heures et moins d'amplitude est de 35 minutes tandis que celle des journées de plus de 8 heures d'amplitude est de 45 minutes. Il est possible de tirer avantage de cette règle pour maximiser le temps utile par rapport au temps payé en construisant des journées de près de 8 heures d'amplitude et ayant une courte pause.

La solution actuelle comporte 53 journées de travail. L'amplitude des journées de travail se situe entre 6h30 et 8h30 et la majorité des pauses ont entre 45 et 59 minutes. Il semble que lors de la fabrication de l'horaire on ait choisi le repas pour être réalisable peu importe l'amplitude de la journée.

Tableau 7.5

Distribution de la durée du repas dans les trois solutions.

MÉTHODE	ACTUELLE	HASTUS	GÉNÉRATION DE COLONNES
Durée du repas	Nombre de Journées de Travail		
35-44	8	24	39
45-59	29	15	10
60-74	9	6	2
75-89	4	2	0
90 et +	3	4	0

Tableau 7.6

Distribution de l'amplitude dans les trois solutions.

MÉTHODE	ACTUELLE	HASTUS	GÉNÉRATION DE COLONNES
Amplitude	Nombre de Journées de Travail		
6h00 et -	0	0	1
6h01-6h30	1	3	5
6h31-7h00	10	4	4
7h01-7h30	11	10	7
7h31-8h00	17	19	26
8h01-8h30	14	15	8

Dans la solution HASTUS pour obtenir une réduction du nombre de conducteurs, l'amplitude moyenne augmente légèrement et la durée des repas diminue. Près de la moitié des journées ont un court repas (moins de 45 minutes), la méthode HASTUS tente de produire des journées efficaces en augmentant légèrement les déplacements. Toutefois, la méthode ne réussit pas à éliminer les très grands repas. Cette solution requiert 51 conducteurs et comporte deux pièces vacantes tôt le matin.

La solution "génération de colonnes" requiert aussi 51 conducteurs et comporte une pièce vacante le matin et une l'après-midi. Trois-quart des journées de travail de cette solution ont un court repas. De plus, la moitié des journées ont une amplitude comprise entre 7h31 et 8h00. La méthode a permis la création de journées efficaces en augmentant les déplacements. Cette méthode a réussi à éliminer complètement les très grands repas (plus de 75 minutes). La méthode de génération de colonnes a permis une amélioration de 1,8% de l'efficacité des journées de travail.

7.5 CONCLUSION

Les premiers essais de la méthode de génération de colonnes montrent que dans les deux problèmes, la méthode a pris avantage de discontinuités dans les coûts. Dans le premier problème, la pause des journées continues est rémunérée entièrement alors que la pause des journées brisées est rémunérée à demi. Il était avantageux de créer des journées brisées pour minimiser les coûts. Dans le second problème, le minimum de durée du repas augmentait de 10 minutes pour les journées de plus de 8 heures. La méthode de génération de colonnes a réussi à produire des journées efficaces.

Ces deux exemples illustrent bien les avantages des méthodes de décomposition du type Dantzig-Wolfe. Le problème maître demande aux sous-problèmes des propositions de journées permettant d'améliorer globalement la solution. Les autres méthodes fonctionnent par amélioration locale de la solution pouvant terminer en un optimum local ou en sélectionnant a priori un sous-ensemble de journées jugées prometteuses pouvant faire partie de la solution. La méthode de génération de colonnes permet une optimisation globale et peut permettre l'obtention d'une solution optimale. Bien que cela soit probablement trop onéreux, il est possible de résoudre optimalement les problèmes en n'effectuant aucune fusion préalable de tâches et en effectuant une exploration complète de l'arbre lors du branchement. Les deux problèmes présentés ont été résolus en utilisant heuristiquement la méthode de génération de colonnes et une amélioration substantielle de la fonction objective est observée par rapport aux méthodes actuelles.

CONCLUSION

Cette thèse apporte une contribution sur trois aspects : une nouvelle modélisation du problème de fabrication des horaires de travail, la résolution des problèmes de plus court chemin avec une ou plusieurs contraintes supplémentaires et la preuve du potentiel d'une méthode de génération de colonnes pour la fabrication d'horaires de travail pour les conducteurs d'autobus par rapport aux méthodes actuelles.

Cette nouvelle modélisation du problème de fabrication d'horaire de travail est définie en terme des paramètres HASTUS utilisés pour décrire les conventions collectives d'une quinzaine de villes. Ceci rend son utilisation pour de nouvelles conventions collectives aisée, puisqu'il suffit généralement de choisir les valeurs des paramètres HASTUS. Toute adaptation supplémentaire ne nécessite que des modifications aux sous-problèmes de plus court chemin avec contraintes supplémentaires (définition du réseau, de coûts ou contraintes de qualification).

En plus de modéliser la fabrication de journées de travail, les problèmes de plus court chemin avec contraintes supplémentaires ont des applications dans le domaine des télécommunications et dans la gestion des réseaux d'ordinateurs [JAFF 84]. L'utilisation de la notion de pulling a permis le design et l'implantation fructueuse d'un algorithme d'un nouveau type pour la résolution de ces problèms. La comparaison de la complexité des algorithmes de pulling et de reaching a permis d'établir la supériorité de l'algorithme de pulling. Ce nouvel algorithme peut être utilisé autant pour résoudre un problème que pour réoptimiser une solution et il permet de résoudre des problèmes de plus court chemin multicritère.

L'obtention de solutions moins coûteuses avec la méthode de génération de colonnes pour les deux problèmes tests a permis de vérifier le potentiel de cette méthode et sa capacité de créer des solutions tirant avantage des possibilités de la convention collective.

Cette méthode présente d'autres avantages. Lors d'une interaction homme-machine sur un problème, il est très facile :

- de fixer une ou des journées de travail;
- de fixer ou d'interdire certaines pauses ou pièces;
- de suggérer des journées désirées (réalisables ou non).

Ces modifications entraînent une réduction de la taille du problème ou des sous-problèmes mais ne posent pas d'entrave à l'obtention d'une solution.

Les tests numériques ont été effectués à l'aide d'une version préliminaire de la méthode de génération de colonnes. Puisque cette version a été réalisée sur un ordinateur CYBER 173 disposant de peu de mémoire centrale, il a été nécessaire, lors de l'implantation, de calculer plusieurs fois certains résultats et d'utiliser la mémoire secondaire pour en conserver d'autres. De plus, la mémoire réduite mise à notre disposition, nous forçait à éliminer certaines colonnes hors-base de la matrice du problème de recouvrement. Parfois ces colonnes ont été générées à nouveau, retardant la convergence de la méthode. Ceci explique partiellement les temps d'exécution considérables rapportés au chapitre précédent. De plus, pour améliorer ces temps d'exécution, il faudra continuer la recherche sur certains aspects de la résolution.

Pour accélérer l'obtention d'une solution initiale, nous avions fixé des paires de tâches. Si cela réduisait beaucoup le temps d'exécution des sous-problèmes, le coût de la solution initiale était relativement élevé. Après examen des solutions entières, une autre possibilité de méthode

heuristique est apparue : rendre plus contraint les paramètres HASTUS portant sur les pauses. En effet, plus le plus court chemin avec contraintes supplémentaires est contraint, plus la résolution des sous-problèmes est rapide.

Dans le premier problème, les pauses peuvent durer jusqu'à 360 minutes alors que la pause la plus longue dans la solution entière est de 156 minutes. Dans le second problème, les pauses peuvent durer jusqu'à 135 minutes alors que la plus longue pause dans la solution entière est de 65 minutes. Dans les deux cas, une diminution de la longueur maximale de la pause n'aurait pas exclu la solution entière, mais aurait permis une diminution considérable du temps de résolution des sous-problèmes. Des contraintes plus fortes sur la longueur des pauses et des pièces de travail pourraient remplacer ou s'ajouter à la première phase suggérée à la section 7.2.

Il faudrait aussi tenter de diminuer le temps de résolution des problèmes de recouvrement. Depuis quelques années, plusieurs méthodes ont été proposées pour la résolution de ces problèmes.

Balas et al. [BALA 80a, BALA 80b] ont proposé une méthode combinant des heuristiques primaux et duaux, des coupes et une méthode de branchement pour obtenir une solution optimale. Récemment, Balas et Ng [BALA 86] ont proposé une autre méthode utilisant la programmation linéaire et des coupes qui sont des facettes du polytope de recouvrement.

Beasley [BEAS 84a] et Kedia [KEDI 85] ont indépendemment mis au point des méthodes utilisant la relaxation lagrangienne et une méthode de branchement pour obtenir la solution optimale de problèmes de recouvrement. Cette approche a permis à Beasley de résoudre des problèmes comptant jusqu'à 300 tâches. Christofides et Paixão [CHRI 84] ont proposé une métho

de semblable combinant la relaxation des états ("State-space relaxation"), la relaxation lagrangienne et une méthode de branchement. Cette méthode a permis de résoudre rapidement des problèmes de 400 tâches. Jaumard [JAUM 86] propose une méthode de résolution de problèmes linéaires en 0-1 par extraction de relations logiques. Cette méthode présente des analogies avec les méthodes utilisées en intelligence artificielle. Cette méthode a l'avantage de pouvoir résoudre des problèmes de recouvrement avec contraintes additionnelles.

Nous pouvons constater que la résolution de problèmes de recouvrement est un domaine de recherche très fécond et qu'il pourrait être possible de s'inspirer de ces nouvelles méthodes pour améliorer notre méthode de résolution des problèmes de recouvrement.

La thèse a permis de confirmer la valeur de la méthode de génération de colonnes pour résoudre des problèmes complexes. Même s'il reste du travail à faire avant de commercialiser un logiciel de fabrication d'horaires de travail pour les conducteurs d'autobus urbains utilisant la méthode de génération de colonnes, l'intérêt pour ce travail est accrû par les résultats de cette thèse.

REMERCIEMENTS

J'aimerais exprimer ma gratitude à mon directeur de recherche, Monsieur François Soumis, professeur agrégé à l'école Polytechnique, pour m'avoir guidé tout au long de cette thèse ainsi que pour l'aide apportée lors de la rédaction d'articles scientifiques et la préparation de communications. Je le remercie tout particulièrement de la confiance et du support financier dont il a fait preuve en me permettant de faire nos communications conjointes lors de plusieurs congrès internationaux.

Aussi, je tiens à remercier Monsieur Jacques Desrosiers, professeur agrégé à l'école des H.E.C. et Monsieur Gilbert Laporte, professeur titulaire à l'école des H.E.C., pour m'avoir fait connaître et aimer les problèmes de routing et scheduling.

Je suis reconnaissant envers Monsieur Jean-Marc Rousseau, directeur du Centre de recherche sur les transports, pour m'avoir suggéré ce sujet de thèse et envers GIRO Inc. et, plus particulièrement, envers Madame Lise Séguin pour m'avoir fourni des données et des statistiques pour me permettre de tester et d'évaluer la méthode proposée dans cette thèse.

J'aimerais remercier Madame Lucie L'Heureux pour l'excellent travail de dactylographie et d'édition de ce texte. Mes remerciements vont aussi à Madame Madeleine Achard Constantin pour son excellent travail lors de la dactylographie de versions préliminaires des chapitres IV, V et VI.

J'aimerais remercier ceux dont le support financier et matériel m'ont permis de terminer rapidement cette thèse : le Conseil de recherche en sciences naturelles et en génie du Canada, le ministère de Transport du Québec, le ministère des Transports du Canada, l'école Polytechnique et le Centre de recherche sur les transports.

Finalement, j'aimerais remercier ceux dont l'amical support m'a permis de traverser ces trois années : les actuels et anciens compagnons de bureau, pour leurs doctorales conversations vipérines, les copains du C.R.T. pour un climat de travail agréable et stimulant et à tous mes amis un remerciement spécial pour tout.

RÉFÉRENCES

[ANEJ 83] ANEJA, Y.P., AGGARWAL, V. et K.P.K. NAIR (1983), "Shortest Chain Subject to Side Constraints", Networks 13, 295-302.

[ARAB 69] ARABEYRE, J.P., FEARNLEY, J., STEIGER, F.C. et W. TEATHER (1969), The Airline Crew Scheduling Problem : A Survey", Transpo tation Science, 294-309.

[BALA 80a] BALAS, E. (1980), "Cutting Planes from Conditional Bounds : a New Approach to Set Covering", Mathematical Programming Study 12, 19-36.

[BALA 80b] BALAS, E. et A. HO (1980), "Set Covering Algorithms Using Cutting Planes, Heuristics, and Subgradient Optimization : A Computational Study", Mathematical Programming Study 12, 37-60.

[BALA 86] BALAS, E. et S.M. NG (1986), "On the Set Covering Polytope : 1, All the Facets with Coefficients in {0,1,2}", Management Science Research Report No. MSSR-522, Carnegie-Mellon University, 28 pages.

[BALL 83] BALL, M., BODIN, L. et R. DIAL (1983), "A Matching Based Heuristics for Scheduling Mass Transit Crews and Vehicles", Transportation Science 17, 4-31.

[BEAS 84a] BEASLEY, J.E. (1984), "An Algorithm for Set Covering Problems", Working paper, Dept. of Management Science, Imperial College, London, 25 pages.

[BEAS 84b] BEASLEY, J.E. et N. CHRISTOFIDES (1984), "An Algorithm for the Resource Constrained Shortest Path Problem", Working paper, Department of Management Science, Imperial College, London, 24 pages.

[BLAI 76] BLAIS, J.-Y., LAPORTE, G., LESSARD, R., ROUSSEAU, J.-M. et F. SOUMIS (1976), "The Problem of Assigning Drivers to Bus Routes in an Urban Transit System", Publication #44, Centre de recherche sur les transports, Université de Montréal, 48 p.

[BODI 83] BODIN, L., GOLDEN, B., ASSAD, A. et M. BALL (1983), "Routing and Scheduling of Vehicles and Crews - The State of the Art", Computer and Operations Research 10, no. 2, special Issue.

[CARP 80] CARPANETO, G. et P. TOTH (1980), "Some New Branching and Bounding Criteria for the Asymmetric Travelling Salemen Problem", Management Science 26, 736-743.

[CARR 82] CARRARESI, P., GALLO, G. et J.-M. ROUSSEAU (1982), "Relaxation Approaches to Large Scale Bus Driver Scheduling Problems", Transportation Research 16B, 383-397.

[CARR 84] CARRARESI, P. et G. GALLO (1984), "Network Models for Vehicles and Crew Scheduling", European Journal of Operational Research 16, 139-151.

[CHRI 84] CHRISTOFIDES, N. et J. PAIXÃO (1984), "Algorithms for Large Scale Set Covering Problems", Working paper, Dept. of Management Science, Imperial College, London, 49 pages.

[CLIM 82] CLIMACO, J.C.N. et E.Q.V. MARTINS (1982), "A Bicriterion Shortest Path Algorithm", European Journal of Operational Research 11, 399-404.

[CRAI 78] CRAINIC, T. (1978), "Le problème des horaires d'équipages d'une compagnie d'aviation", publication #122, Centre de recherche sur les transports, Université de Montréal.

[CRAI 86] CRAINIC, T. et ROUSSEAU, J.M. (1986), "The Column Generation Principle and the Airline Crew Scheduling Problem", publication #241, Centre de recherche sur les transports, Université de Montréal, 21 pages.

[CULL 81] CULLEN, F., JARVIS, J. et H.D. RATLIFF (1981), "Set Partitioning Based Heuristics for Interactive Routing", Networks 11, No 2, 125-144.

[CULL 84] CULLEN, F. (1984), "Set Partitioning Based Heuristics for Interactive Routing", Ph.D. thesis, Georgia Institute of Technology.

[DANT 60] DANTZIG, G.B. et P. WOLFE (1960), "The Decomposition Algorithm for Linear Programming", Operations Research 8.

[DENA 79] DENARDO, E.V. et B.L. FOX (1979), "Shortest-Route Methods : 1. Reaching, Pruning and Buckets", Operations Research 27, 161-196.

[DESR 83] DESROSIERS, J., PELLETIER, P. et F. SOUMIS (1983), "Plus court chemin avec contraintes d'horaires", RAIRO Recherche opérationnelle 17, 357-377.

[DESR 84a] DESROSIERS, J., SOUMIS, F. et M. DESROCHERS (1984), "Routing with Time Windows by Column Generation", Networks 14, 545-565.

[DESR 84b] DESROSIERS, J. DUMAS, Y. et F. SOUMIS (1984), "The Multiple Vehicles Many to Many Routing Problem with Time Windows", publication #362, Centre de recherche sur les transports, Université de Montréal.

[DION 78] DIONNE, R. (1978), "Étude et extension d'un algorithme de Murchland", INFOR 16, 132-146.

[DUMA 85] DUMAS, Y. (1985), "Confection de routes de véhicules pour le transport de plusieurs origines à plusieurs destinations", publication #434, Centre de recherche sur les transports, Université de Montréal.

[FUJI 81] FUJISHIGE, S. (1981), "A Note on the Problem of Updating Shortest Paths", Networks 11, 317-319.

[GALL 80] GALLO, G. (1980), "Reoptimization Procedures in Shortest Path Problems", Rivista di matematica e di Science economiche e Sociali 3, 1, 3-13.

[GALL 86] GALLO, G. et S. PALLOTTINO (1986), "Shortest Paths Method : A Unifying Approach", Math. Prog. Study 26, 38-63.

[GILM 61] GILMORE, P.C. et R.E. GOMORY (1961), "A Linear Programming Approach to Cutting-Stock Problem", Operations Research 9, 849-859.

[GILM 63] GILMORE, P.C. et R.E. GOMORY (1963), "A Linear Programming Approach to Cutting-Stock Problem - part II", Operations Research, 863-888.

[GIRO 83] GIRO Inc., (1983), "Manuel de l'utilisateur d'HASTUS-MACRO, version 2.2", Montréal.

[GOTO 78] GOTO, A. et A. SANGIOVANNI-VINCENTELLI (1978), "A New Shortest Path Updating Algorithm", Networks 8, 341-372.

[HABE 81] HABENICHT, W. (1981), "Efficient Routes in Vector-Values Graphs", Proceedings of the Seventh Conference on Graph-theoretic Concepts in Computer Science, J.R. Mühlbacher, ed., Carl Hanser Verlag, Munich, 349-355.

[HAJJ 81] HAJJ, I.N. (1981), "An Updating Algorithm for the Shortest Path Problem", Proceedings of the IEEE International Symposium on Circuits and Systems, 27-29 avril 1981, Chicago, 802-805.

[HAND 80] HANDLER, G.Y. et I. ZANG (1980), "A Dual Algorithm for the Constrained Shortest Path Problem", Networks 10, 293-310.

[HANS 80] HANSEN, P. (1980), "Bicriterion Path Problems, dans G. Fandel et T. Gal, éds., Multiple Criteria Decision Making : Theory and Applications, Lecture Notes in Economics and Mathematical Systems 177, Springer Verlag, Heidelberg, 109-127.

[HEUR 72] HEURGON, E. (1972), "Un problème de recouvrement : l'habillage des horaires d'une ligne d'autobus", Revue d'Automatique, d'Informatique et de Recherche Opérationnelle 6, 13-29.

[HEUR 75] HEURGON, E. (1975), "Preparing Duty Roster for Bus Routes by Computer", Preprint workshop on automated technique for scheduling of vehicle operators for urban public transportation services, Bergmann et Bodin Eds., Chicago.

[JAFF 84] JAFFE, J.M. (1984), "Algorithms for Finding Paths with Multiple Constraints", Networks 14, 95-116.

[JAUM 86] JAUMARD, B. (1986), "Extraction et utilisation des relations booléennes en programmation linéaire 0-1", thèse de doctorat, Ecole Nationale Supérieure des Télécommunications, Paris, 450 pages.

[JOKS 66] JOKSCH, H.C. (1966), "The Shortest Route Problem with Constraints", Journal of Mathematical Analysis and Applications 14, 191-197.

[KEDI 85] KEDIA, P.K. (1985), "Lagrangian Multiplier Adjustment Method for Solving Certain Combinatorial Problems", Ph.D. Thesis, Wharton School, University of Pennsylvania, 102 pages.

[KUNG 75] KUNG, H.T., LUCCIO, F. et F.P. PREPARATA (1975), "On Finding the Maxima of a Set of Vectors", JACM 22, 5, 469-476.

[LAVO 85] LAVOIE, S., MINOUX, M. et E. ODIER (1985), "A New Approach of Crew Pairing Problems by Column Generation and Application to Air Transport", rapport interne, Centre de recherche sur les transports, Université de Montréal, 26 p.

[LEMK 71] LEMKE, C., SALKIN, H. et K. SPIELBERG (1971), "Set Covering by Single Branch Enumeration with Linear Programming Subproblems", Operations Research 19, 998-1022.

[LESS 81] LESSARD, R., ROUSSEAU, J.-M. et D. DUPUIS (1981), "HASTUS I : A Mathematical Programming Approach to the Bus Driver Scheduling Problem", dans [WREN 81], 255-267.

[MANI 75] MANINGTON, B. et A. WREN (1975), "A General Method for Bus Crew Scheduling", Preprint workshop on automated technique for scheduling of vehicle operators for urban public transportation services, Bergmann et Bodin Eds., Chicago.

[MANI 77] MANINGTON, P.D. (1977), "Mathematical and Heuristics Approaches to Road Transport Scheduling", Ph.D. thesis, University of Leeds.

[MARS 81a] MARSTEN, R.E. et F. SHEPARDSON (1981), "Exact Solution of Crew Scheduling Problems Using the Set Partitionning Model : Recent Successful Applications", Networks 11, 165-177.

[MARS 81b] MARSTEN, R.E. (1981), "The Design of the XMP Linear Programming Library", ACM Transcation on Mathematical Software 7, 481-497.

[MART 84] MARTINS, E.Q.V. (1984), "On a Multicriteria Shortest Path Problem", European Journal of Operational Research 16, 236-245.

[MINO 75] MINOUX, M. (1975), "Plus court chemin avec contraintes : algorithmes et applications", Annales des Télécommunications 30, 12, 1-12.

[MITR 85] MITRA, G. et K. DARBY-DOWNMAN (1985), "CRU-SHED : A Computer Based Bus Crew Scheduling System Using Integer Programming", in [ROUS 85a], 223-232.

[MURC 67] MURCHLAND, J.D. (1967), "The Effect of Increasing and Decreasing the Length of a Single Arc All Shortest Distances in a Graph", Report LBS-TNT-26, London Graduate School of Business Studies.

[PAPE 74] PAPE, U. (1974), "Changes in Networks and Adjustment of the Lengths of Shortest Paths", Computing 12, 357-362.

[RIBE 83] RIBEIRO, C.C.C. (1983), "Algorithmes de recherche de plus courts chemins avec contraintes : étude théorique, implémentation et parallélisation. Thèse présentée à l'École Nationale Supérieure des Télécommunications, Paris, 237 pages.

[RODI 68] RODIONOV, V. (1968), "The Parametric Problem of Shortest Distances", Urdmurt. U.S.S.R. Computat. Math. Math Phys. 8, 336-342.

[ROUS 84] ROUSSEAU, J.-M. (1984), "Crew Scheduling Methods in the Operation and Planning of Transportation Systems", dans Transportation Planning Models", M. Florian Editor, Elsevier Science Publishers, Amsterdam, 439-472.

[ROUS 85a] ROUSSEAU, J.-M. Éditeur (1985), "Computer Scheduling of Public Transport 2", Elsevier Science Publishers, Amsterdam.

[ROUS 85b] ROUSSEAU, J.-M., LESSARD, R. et J.-Y. BLAIS (1985), "Enhancement to the HASTUS Crew Scheduling Algorithm", in [ROUS 85a], 295-310.

[RYAN 81] RYAN, D.M. et B.A. FORSTER (1981), "An Integer Programming Approach to Scheduling", in [WREN 81], 269-280.

[SHEP 80] SHEPARDSON, F. et R.E. MARSTEN (1980), "A Lagrangian Relaxation Algorithm for the Two Duty Period Scheduling Problem", Management Science 26, 274-281.

[SLEA 83] SLEATOR, D.D. et R.E. TARJAN (1983), "Self-Adjusting Binary Trees", Proc. Fifteenth Annual ACM Symposium on the Theory of Computing, 235-245.

[SPIR 75] SPIRA, P.M. et A. PAN (1975), "On Finding and Updating Spanning Trees and Shortest Paths", SIAM Journal on Computing 4, 375-380.

[VINC 74] VINCKE, P. (1974), "Problèmes multicritères", Cahiers du Centre d'Études en Recherche Opérationnelle 16, 425-439.

[WARB 86] WARBURTON, A. (1986), "Approximation of Pareto Optima in Multiple Objective Shortest Path Problems", Working paper, Faculty of Management Science, University of Ottawa, 36 pages.

[WARD 81] WARD, R.E., DURANT, P.A. et A.B. HALLMAN (1981), "A Problem Decomposition Approach to Scheduling the Drivers and Crews of Mass Transit Systems", in [WREN 81], 297-312.

[WILH 75] WILHELM, E.B. (1975), "Overview of the RUCUS Package Driver Run Cutting Program (RUNS)", Preprint workshop on automated technique for scheduling of vehicle operators for urban public transportation services, Bergmann et Bodin Eds., Chicago.

[WREN 81] WREN, A. Éditeur (1981), "Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling", Elsevier Science Publishers, Amsterdam.

[WREN 85] WREN, A., SMITH, B.M. et A.J. MILLER (1985), "Complementary Approaches to Crew Scheduling", in [ROUS 85a], 263-278.

[YUCA 73] YUCAOGLU, E. (1973), "Solution Algorithms for Resource and Route Constrained Shortest-Path Problems in Time-Dependant Transportation Networks", Ph.D. Thesis, Stanford University.